UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

TRABAJO FIN DE GRADO

Recomendación de contactos en Twitter

Alfonso Alhambra Morón
Tutor: Pablo Castells Azpilicueta

Mayo 2014

Resumen

En el presente trabajo de fin de grado abordamos la recomendación en redes sociales centrándonos en el caso particular de la recomendación de usuarios en *Twitter*, una red social que presenta acceso abierto y diversidad de estructuras, tipos de elementos y conexiones.

Centramos nuestro estudio en la adaptación de algoritmos típicos del ámbito de la recuperación de información basados en grafos tales como *PageRank* o *HITS* así como algoritmos más orientados a la clasificación basada en contenido como *Rocchio*. También nos hemos aproximado al problema de la combinación de rankings para aprovechar de forma conjunta dos o más algoritmos distintos a la hora de generar recomendaciones.

El trabajo realizado abarca desde la preparación de un conjunto de datos experimental compuesto por aproximadamente 10.000 usuarios reales y más de 2.000.000 *tweets* hasta la evaluación de distintos algoritmos de recomendación de usuarios haciendo uso de métricas de precisión, novedad y diversidad.

Dentro de este proceso se ha desarrollado una plataforma de experimentación que permite la preparación de conjuntos de datos de *Twitter* con distintas configuraciones, su partición en conjuntos de entrenamiento y test y la implementación y ejecución sistematizada de distintos algoritmos de recomendación y métricas de evaluación. Esta plataforma se ha diseñado buscando la máxima flexibilidad y es compatible con distintas aproximaciones a la recomendación en *Twitter* incluyendo, por ejemplo, la recomendación de *tweets*.

El trabajo realizado ha contribuido a ampliar los recursos para investigación del grupo de Recuperación de Información. En particular, a la fecha presente, el conjunto de datos preparado en este trabajo ha sido utilizado en otros dos trabajos de fin de grado, en una de las prácticas de la asignatura de búsqueda y minería de información y en la publicación de un póster acerca de novedad y diversidad en recomendación en redes sociales en el III Congreso Español de Recuperación de Información (CERI 2014) que se celebrará en A Coruña en Junio de este año.

Palabras clave: Recomendación, red social, *Twitter*, conjuntos de datos, novedad, diversidad.

Abstract

In this bachelor thesis we approach the topic of recommendation systems in the context of social networks. In particular we focus in user recommendations in *Twitter*, a social network which presents open access to data and various structures, item types and connections.

We have lined up to the adaptation of graph-based algorithms typically related to the area of information retrieval such as *PageRank* or *HITS* as well as classification-oriented algorithms such as *Rocchio*. We have also approached the problem of rankings aggregation to be able to jointly exploit two or more recommendation algorithms when generating recommendations.

The accomplished work ranges from the preparation of a set of experimental data consisting of approximately 10,000 real users and more than 2,000,000 tweets to evaluating various user recommendation algorithms making use of precision, novelty and diversity metrics.

Within this process we have developed an experimental platform which allows the preparation of *Twitter* datasets with different settings, the datasets partition into training and test datasets and the systematic implementation and execution of different recommendation algorithm and evaluation metrics. This platform has been designed towards maximizing flexibility and it supports different approaches to recommendation in *Twitter* including, for example, *tweets* recommendation.

The accomplished work has helped expanding the Information Retrieval Group's research resources. In particular, so far, the set of prepared experimental data has been used in other two bachelor thesis as well as in one of the projects in the subject information search and mining. It has also been used as the key data source in a poster about novelty and diversity in social networks recommendation systems published in the III Spanish Conference of Information Retrieval (CERI 2014) to be hold in A Coruña in June this year.

Keywords: Recommender systems, social network, *Twitter*, datasets, novelty, diversity.

Agradecimientos

La realización de este trabajo no habría sido posible sin la ayuda, el apoyo y sobre todo la comprensión que me han brindado todas las personas que menciono en esta pequeña página de agradecimientos que se queda corta para expresar mi gratitud hacia todos.

Quisiera empezar agradeciendo a Pablo, mi tutor, todas las horas dedicadas a enseñarme todo lo necesario para realizar este trabajo, pensando conmigo para resolver los distintos problemas que se han planteado en la realización de este trabajo de fin de grado y leyendo y releyendo la memoria y la presentación para cuidar hasta el último detalle y hasta el último momento de la entrega. Aprovecho para agradecer su apoyo y su motivación así como su respaldo de cara a mis estudios futuros respaldando mi candidatura a todos los másters y becas solicitados. Gracias por todo, Pablo.

No puedo seguir por otra persona que no sea Alma, que me ha acompañado en los mejores y en los peores momentos de este trabajo aconsejándome siempre en la mejor línea posible y dándome todo el apoyo, tanto intelectual como sentimental, necesario para terminar este trabajo. Después de tantos ensayos y llegando a aprenderse de memoria mi presentación del mismo, he de reconocer que el mérito es casi más suyo que mío. Gracias de corazón, Alma.

También estoy agradecido a mis compañeros de laboratorio a lo largo de este año: a Saúl, especialmente por aquellas charlas acerca de diferenciar predicción de rating de asignación de score que tanto me ayudaron a aclarar conceptos. A Víctor, por no rechistar nunca cuando le hacía darse la vuelta con cualquier pregunta sobre Twitter, a Rocío y a Sofía, compañeras de tutela, que tan buenos consejos me han dado y a todos los demás, mil gracias.

Para terminar, me gustaría agradecer su apoyo y cariño a mi familia, amigos y profesores. Aquellos que se han preocupado por cómo avanzaba mi trabajo y que se han alegrado con su desenlace se pueden ver identificados en estas líneas. Mencionar especialmente a mi padre, que llegó a entender la esencia del trabajo y me dio muchas fuerzas y consejo en alguno de los momentos más críticos y a mi madre y mis hermanas, que no dudaron en aguantar en silencio el cuento en un idioma extraño que representaba mi trabajo de fin de grado sabiendo valorar la importancia que para mí tenía.

Mil gracias a todos y por todo.

Contenido

Índice de tablas

Índice de figuras

1. Introducción

1.1 Motivación

La recomendación está cada vez más presente en tecnología de uso diario relacionada con el acceso a la información y la selección de opciones entre conjuntos masivos de alternativas. La proliferación de los distintos sistemas de venta online o repositorios de información especializada fomentan el uso de la tecnología de recomendación al crear cada vez más escenarios en los que el usuario tiene acceso a cantidades ingentes de información y el proceso de discriminación de la información relevante para el usuario se convierte en una tarea inabarcable sin ayuda. Para facilitar una solución a este problema, los sistemas de recomendación observan la interacción entre los usuarios y los objetos (ítems) susceptibles de ser recomendados, detectan patrones de preferencia en estas interacciones y, en base a ellas, generan una lista ordenada (generalmente personalizada) de ítems a recomendar a cada usuario.

Hoy en día podemos encontrar sistemas de recomendación integrados en páginas web frecuentemente utilizadas como es el caso de Youtube, que nos recomienda los vídeos más populares de los canales que seguimos así como los videos más afines al contenido que solemos consumir o los más valorados por nuestros contactos en Google. Otro ejemplo muy popular es el de Amazon que nos recomienda artículos de consumo en función de las valoraciones que asignemos a distintos productos con los que comercian o los productos que nosotros mismos hemos consumido.

Desde principios de los noventa se ha desarrollado una amplia multitud de modelos, algoritmos y metodologías de evaluación en el área de la recomendación que se basan típicamente en una representación simplificada de la interacción entre usuarios e ítems (el input para el sistema de recomendación) consistente en un valor numérico (o en ocasiones binario) que refleja un grado de interés explícitamente expresado por el usuario por una serie de ítems.

Por otro lado, desde principios de los 2000 estamos asistiendo al auge masivo de las redes sociales online. Y las funcionalidades de recomendación confluyen con ellas con sentido pleno: recomendar usuarios, utilizar estructura y trazas propias de las redes sociales como dato de entrada para mejorar las recomendaciones, etc. En estos nuevos escenarios se nos plantea un nuevo problema que conlleva un giro conceptual en el ámbito de los sistemas de recomendación: Los usuarios no expresan sus preferencias de forma estructurada; por ejemplo, un usuario de *Facebook* no valora a otros usuarios asignándoles 5 estrellas o una estrella, un usuario de *Twitter* no va a mostrar una preferencia estable por un tema de interés temoporal como puede ser un festival de Eurovisión o un partido de fútbol. En este sentido es necesario capacitar a los sistemas de recomendación para generar un modelo de preferencias del usuario de forma implícita y para tener en cuenta variables como el tiempo a la hora de generar recomendaciones.

Este trabajo fin de grado se desarrolla en este ámbito, en concreto se centra en el ámbito de *Twitter*, que se presta muy especialmente a este tipo de investigación y experiencia por su acceso abierto y la riqueza de sus estructuras, tipos de elementos y conexiones. El trabajo se ha orientado a la definición, implementación y prueba de algoritmos de recomendación específicamente dirigidos a entornos de red social. Parte del problema a tratar en el trabajo que aquí se documenta es la obtención y preparación de datos para llevar a cabo la experiencia. También se ha abordado la adaptación de distintos algoritmos para la recomendación de usuarios tomando como fuente de datos las interacciones sociales observadas entre distintos usuarios y los *tweets* publicados por los mismos.

1.2 Objetivos

Las líneas a las que se orienta este trabajo se han materializaco en los siguientes objetivos concretos:

- Preparar un conjunto de datos de *Twitter* adecuado para experimentar con funcionalidades de recomendación. Este objetivo no se limita al ámbito de este trabajo de fin de grado, donde nos vamos a centrar en la recomendación de usuarios, sino que abarca el diseño y la implementación de un sistema que permita la preparación automatizada de conjuntos de datos de *Twitter* aptos para la recomendación de usuarios, *tweets* o incluso noticias o páginas web.
- Desarrollar una plataforma que permita la experimentación y prueba sistemática de algoritmos de recomendación. Esta plataforma va a dar cobertura a la experimentación de recomendación en *Twitter* desde la preparación del conjunto de datos experimental hasta la producción final de los reportes de evaluaciones de algoritmos de recomendación pasando por la separación del conjunto de datos en conjuntos de entrenamiento y test, la indexación de los *tweets* de entrenamiento y su tratamiento para facilitar la recomendación basada en su contenido, la generación de listas de ítems recomendados para los usuarios haciendo uso de distintos algoritmos de recomendación y la interacción con herramientas externas de evaluación de listas de recomendación. Al igual que sucedía con el objetivo anterior, esta plataforma va más allá de la recomendación de usuarios únicamente siendo su diseño compatible con algoritmos de recomendación de otros tipos de ítems tales como *tweets* o noticias.
- Definir e implementar algoritmos de recomendación específicos de redes sociales. En particular nos vamos a enfocar en algoritmos específicos de recomendación de usuarios en *Twitter*. Dentro de este ámbito vamos a aproximarnos a algoritmos de dos tipos: los basados en grafos, que utilizarán como input el grafo de red social representativo de las interacciones observadas entre los usuarios en el conjunto de datos de entrenamiento y los basados en contenido, que utilizarán como input los *tweets* publicados por los usuarios en la red social. Para ampliar las posibilidades de aprovechamiento de estos algoritmos vamos a implementar un algoritmo de hibridación que permita utilizar dos o más algoritmos de recomendación de usuarios conjuntamente.

- Probar y comparar los algoritmos en diferentes configuraciones y condiciones de prueba. En particular se evaluará la precisión de los algoritmos de recomendación así como su novedad y diversidad. Para permitir la evaluación se separará el conjunto de datos preparado en un conjunto de entrenamiento que será utilizado para generar recomendaciones y en un conjunto de test que será utilizado para evaluar las recomendaciones generadas.

1.3 Estructura del trabajo

En el capítulo 2 introducimos brevemente el contexto y trabajo relacionado más relevante para el trabajo presentado analizando en particular el contexto de los sistemas de recomendación, las redes sociales y la recomendación en redes sociales.

En el capítulo 3 presentamos el conjunto de datos preparado conforme al primero de los objetivos presentados describiendo todos los aspectos relacionados con su construcción así como su estructura y procedemos finalmente a un breve análisis del mismo.

En el capítulo 4 describimos en detalle la plataforma de experimentación diseñada presentando el diagrama detallado de flujo de datos en la misma desde la recuperación de información de *Twitter* hasta la generación de ficheros de evaluación de los algoritmos de recomendación.

En el capítulo 5 presentamos y describimos en detalle todos los algoritmos de recomendación implementados que han sido utilizados en la realización de los experimentos asociados a este trabajo de fin de grado. En este capítulo vamos a encontrar para cada algoritmo su formulación matemática así como una representación simplificada de su funcionamiento.

En el capítulo 6 describimos en detalle la configuración de todos los experimentos llevados a cabo incluyendo la descripción detallada de todas las métricas empleadas en la evaluación de los algoritmos de recomendación. Hecho esto presentamos de forma agregada los resultados más relevantes obtenidos.

Por último en el capítulo 7 procedemos a una breve recapitulación de los objetivos alcanzados durante la realización del TFG así como las contribuciones más interesantes y establecemos posibles líneas de trabajo futuro que parten de la plataforma desarrollada y los experimentos realizados.

2. Antecedentes

Antes de presentar el desarrollo de nuestro trabajo en la consecución de los objetivos planteados en el apartado 0 introducimos aquí brevemente el contexto y trabajo relacionado más relevante para el trabajo realizado. Ello incluye sistemas de recomendación, redes sociales y sistemas de recomendación en redes sociales.

2.1 Sistemas de recomendación

Los sistemas de recomendación son herramientas de software y técnicas para la sugerencia de ítems relevantes para un usuario. ”Ítem” es el término generalmente utilizado para designar lo que el sistema recomienda al usuario y se puede referir a documentos, vídeos, bienes de consumo o, como en el caso particular de este trabajo de fin de grado, otros usuarios. En su forma más simple, la tarea de recomendación consiste en ofrecer al usuario una lista ordenada de ítems. Para ello, el sistema de recomendación intenta predecir cuáles son los ítems más relevantes para un usuario en base a determinadas preferencias evidenciadas expresa o implícitamente por el usuario. Para ordenar los ítems por relevancia para un usuario, el sistema de recomendación recupera las preferencias de un usuario que o bien están expresadas de forma explícita, por ejemplo como puntuaciones asignadas por el usuario a distintos ítems, o bien han de ser inferidas interpretando las acciones de un usuario (Ricci et al 2011). En este sentido podemos entender un algoritmo de recomendación como un sistema que dado un usuario y un ítem susceptible de ser recomendado al mismo, asigna al ítem un *score* que permite situarlo en la posición oportuna de la lista ordenada de ítems recomendados que el sistema ofrece al usuario.

Los algoritmos de recomendación más comunes se clasifican habitualmente dentro de tres categorías principales: basados en contenido, filtrado colaborativo y algoritmos híbridos (Adomavicius & Tuzhilin, 2005). Entre los algoritmos colaborativos destacan entre otros muchos el algoritmo *kNN* basado en usuario (*k Nearest Neighbors*), que se inspira en el algoritmo de clasificación supervisada con el mismo nombre (Fix & Hodges, 1951). La recomendación por kNN basado en usuario establece el *score* que se asigna a un ítem susceptible de ser recomendado a un usuario como el promedio ponderado de las valoraciones explícitas que hayan asignado los *k* usuarios más parecidos al usuario (dentro del conjunto de usuarios que hayan valorado el ítem), donde la ponderación se basa en una medida de similitud entre el usuario objetivo y sus vecinos. También es digno de mención entre los algoritmos de filtrado colaborativo el algoritmo de factorización de matrices (Koren, et al., 2009) orientado a la recomendación de productos adaptado a la introducción de información de recomendación adicional tal como feedback implícito, efectos temporales y niveles de confianza.

Entre los algoritmos basados en contenido destaca la variante del algoritmo *kNN* basada en ítem, que establece el *score* que se asigna a un ítem como el promedio de la

valoración explícita que el usuario haya asignado a los *k* ítems más similares (dentro de los ítems valorados por el mismo) al ítem susceptible de ser recomendado. También es digno de mención el algoritmo Rocchio (1971) que establece el *score* de un ítem susceptible de ser recomendado a un usuario como el promedio de las valoraciones asignadas por los distintos usuarios a ese ítem ponderado por la similitud de los usuarios con el usuario susceptible de recibir recomendaciones. Este algoritmo en particular es relevante en el contexto de nuestro trabajo de fin de grado ya que es el algoritmo que va constituir la base de nuestra aproximación a la utilización de los *tweets* publicados por los usuarios para generar recomendaciones de usuarios basadas en contenido.

Por último los algoritmos híbridos combinan sendos enfoques ya sea a nivel específico combinando a nivel de algoritmo inputs diversos o a un nivel menos específico pero más flexible combinando las listas ordenadas generados por distintos algoritmos de recomendación. Es en la línea del enfoque de combinación de rankings en la que vamos a abordar en nuestro trabajo de fin de grado la hibridación de algoritmos de recomendación.

2.2 Redes sociales

Entendemos formalmente las redes sociales como una estructura social compuesta por un conjunto de actores sociales (ya sean individuos u organizaciones) y un conjunto de relaciones o interacciones entre pares de esos actores (Wasserman & Faust, 1994).

Los estudiosos se han empezado a fijar en los fenómenos en red a principios del siglo XX, con el tiempo se han sumado investigadores de diversas disciplinas científicas (matemáticas, física), con avances teóricos muy notables. Finalmente el surgimiento de las redes online abre una nueva dimensión en este campo de estudio por la posibilidad de visualizar las redes y experimentar con datos reales de escala sin precedentes (Russell, 2011). Los temas de estudio en este campo abarcan el análisis y la modelización de redes en cuanto a su estructura mediante métricas y otras estadísticas y propiedades topológicas de nivel micro, meso y macro; así como fenómenos como la propagación de información y estados.

En el trabajo que aquí se presenta las redes sociales son un contexto para el desarrollo del mismo. No abordaremos cuestiones de análisis profundo de las redes en sí, pero sí se tomará nota de ciertas características básicas de la red que se va a obtener y sobre la que se va a trabajar. Además exploraremos la utilización de métricas de análisis de red social como PageRank (Page & Brin, 1998), HITS (Kleinberg, 1999) o Betweenness, como elementos base para la definición de algoritmos de recomendación.

2.3 Recomendación en redes sociales

En los últimos tiempos la recomendación ha empezado a establecerse como una herramienta de uso común en la mayoría de las redes sociales. En estos escenarios aparecen nuevos retos para los sistemas de recomendación tales como la ausencia de feedback explícito que permita utilizar de forma directa los distintos sistemas de recomendación.

Además aparecen diversos tipos de elementos susceptibles de ser recomendados incluyendo a los propios usuarios, que entran a jugar el doble papel de usuario e ítem. A día de hoy nos encontramos en una frase temprana del estudio estructurado y sistematizado de la recomendación en redes sociales y el trabajo realizado en estos ámbitos típicamente se centra en tareas de recomendación individuales. En este sentido la recomendación basada en redes sociales ofrece un escenario amplio y atractivo de investigación en el que se dispone de bastante libertad para investigar en distintas direcciones.

En particular en el caso de la red social *Twitter* encontramos ya las primeras iniciativas de aproximación a un estudio estructurado de sistemas de recomendación de red social, un buen ejemplo es (Kywe et al, 2012), que propone enfoques de recomendación en red social considerando desde los tradicionales sistemas de filtrado colaborativo y basados en contenido, hasta enfoques más adaptados al escenario de las redes sociales como lo son aquellos basados en la topología de red social. También se mencionan sistemas basados en contenido ponderados o métodos estructurales y se toman en consideración enfoques típicos de distintas áreas de la recuperación de información tales como la representación vectorial de usuarios mediante el modelo vectorial *tf-idf*, utilizando para ello los *hashtags* más relevantes para los usuarios. Este último enfoque está íntimamente relacionado con los experimentos llevados a cabo en este trabajo de fin de grado ya que se han representado usuarios mediante el modelo *tf-idf* con la diferencia de que en nuestro caso hemos optado por utilizar todo el contenido de los *tweets* como información contextual en lugar de únicamente los *hashtags*.

3. Conjunto de datos

Para realizar recomendaciones basadas en red social, y como primera fase del trabajo que aquí se presenta, hemos reunido y preparado un conjunto de datos extraído de la red social *Twitter*, que incluye un subconjunto de los datos más recientes de un total de 10.029 usuarios con perfil público en la red social.

Dadas las dimensiones de la red social de *Twitter* es naturalmente inabarcable utilizar la información de todos los usuarios de la red social para realizar experimentos, por este motivo es necesario seleccionar un pequeño subconjunto de los usuarios que sea representativo de la red social. La selección de este subconjunto de usuarios no es un problema trivial, es necesario asegurarnos de recuperar todas las interacciones entre usuarios del conjunto de datos final sin conocer a priori cuales van a ser los usuarios finales y es también necesario filtrar mínimamente los usuarios para evitar la inclusión en el conjunto de datos de usuarios que puedan distorsionar los experimentos (como por ejemplo usuarios completamente inactivos o usuarios que siguen a miles o millones de usuarios y realmente son cuentas de generación de spam). Además es importante tener en cuenta las fuertes restricciones en el acceso a datos que presenta *Twitter* lo cual exige la optimización de cualquier herramienta de recuperación de información para evitar solicitudes redundantes de información y aprovechar al máximo el ancho de banda ofrecido.

Vamos a comenzar este capítulo con una primera descripción breve de las herramientas desarrolladas para la preparación del conjunto de datos, los conceptos más relevantes asociados a las mismas y una ilustración progresiva del funcionamiento de las mismas a lo largo de la cual irán apareciendo los problemas clave asociados a la recuperación de información de *Twitter* de forma progresiva. A continuación vamos a presentar las decisiones de diseño y detalles técnicos más relevantes de las herramientas de presentadas. Para terminar vamos a presentar la estructura del conjunto de datos y procedemos a un breve análisis del conjunto de datos preparado.

3.1 Introducción a la preparación del conjunto de datos

A la hora de preparar el conjunto de datos uno de los primeros detalles que tenemos que tener en cuenta es que nos tenemos que anticipar a la decisión de cuales van a ser los usuarios finales del conjunto de datos a medida que vamos recolectando información de usuarios para la preparación del conjunto de datos. Esto es muy relevante ya que determina las dos fases principales de nuestro proceso de preparación del conjunto de datos: La fase de recuperación de información (en la que recuperaremos más información de la estrictamente necesaria para anticiparnos a la inserción de nuevos usuarios finales y evitar repetir solicitudes de la misma información a *Twitter*) y la fase de limpieza del conjunto de datos final (en la que eliminamos del conjunto de datos final la información sobrante).

Para llevar a cabo las tareas asociadas a las dos fase de preparación del conjunto de datos presentadas hemos diseñado e implementado respectivamente dos programas, *Crawler* y *DbIsolator*, a continuación presentamos los conceptos básicos que vamos a tomar en consideración a la hora de construir el conjunto de datos e ilustramos el funcionamiento de sendos programas simulando la preparación de un pequeño conjunto de datos.

3.1.1 Conceptos básicos en la preparación del conjunto de datos

Para entender el funcionamiento de los programas *Crawler* y *DbIsolator* conviene establecer los conceptos que se van a ver involucrados en el proceso de preparación del conjunto de datos.

API REST de *Twitter*: Es la interfaz de programación de aplicaciones ofrecida por *Twitter* que permite la descarga de información mediante peticiones HTTP GET. Esta información se ofrece en formato JSON (formato ligero para el intercambio de datos basado en la notación literal de objetos en JavaScript). El programa *Crawler* almacena esta información por una parte en un repositorio JSON para evitar repetir innecesariamente llamadas a la API y dejar abierta la puerta a la ampliación futura de la información contextual asociada al conjunto de datos preparado sin necesidad de recurrir de nuevo a la API y por otra parte almacena un subconjunto de la información ofrecida en una base de datos MySQL estructurada cuya estructura se describe en el apartado 3.3.

Tipos de usuario: Se consideran tres tipos de usuarios finales en el conjunto de datos en función de su influencia en la construcción del mismo:

- **Usuario raíz**: Es el primer usuario que se introduce en el conjunto de datos.
- **Usuarios rama**: Son los usuarios cuyos *followees* (usuarios que ellos siguen) van a ser introducidos como usuarios finales del conjunto de datos final.
- **Usuarios hoja**: Son los usuarios cuyos *followees* no van a ser introducidos como usuarios finales del conjunto de datos final.

Expansión: Recuperación de la lista de identificadores de los *followees* de un usuario del conjunto de datos y, en caso de que el usuario expandido sea *raíz* o *rama*, recuperación de la información de usuario asociada a sus *followees* e inserción de éstos en el conjunto de datos. Es importante tener en cuenta que en el momento de su expansión, si ya se ha alcanzado el número mínimo de usuarios finales del conjunto de datos, los usuarios *rama* van a ser transformados en usuarios *hoja* permitiendo así que el proceso de recuperación de información finalice sin necesidad de recuperar todos los usuarios de *Twitter*.

Estados de expansión de un usuario: Se diferencian cuatro estados de expansión de un usuario durante la preparación del conjunto de datos de los cuales sólo uno permanecerá en el conjunto de datos tras la ejecución de los programas *Crawler* y *DbIsolator*:

- **Para expandir**: Usuario que ha sido insertado en el conjunto de datos como usuario final pero aún no ha sido expandido. Esta inserción puede haberse producido por las siguientes razones:
    - Por ser el usuario *raíz*.

    - Por ser seguido por el usuario *raíz* o un usuario *rama* y haber sido aprobado como usuario final del mismo por superar los criterios de aceptación de usuarios que configuran el *Crawler*.
- **En expansión**: Usuario que ha sido insertado en el conjunto de datos como usuario final y está siendo expandido. Este estado permite la sincronización entre distintas instancias del programa *Crawler* que colaboren en la construcción de un mismo conjunto de datos.
- **Expandido**: Usuario que ha sido insertado en el conjunto de datos como usuario final y ya ha sido expandido. Este es el único estado de expansión que podrá ser observado en los usuarios finales del conjunto de datos tras la ejecución completa de los programas *Crawler* y *DbIsolator*. En particular todos los usuarios cuyo estado de expansión tras la finalización del proceso de *crawling* no sea éste serán eliminados del conjunto de datos final por el programa *DbIsolator*.
- **No expandible**: Usuario que ha sido insertado provisionalmente en el conjunto de datos pero no ha sido aprobado como usuario final del mismo. Esta inserción puede haberse producido por las siguientes razones:
    - Ser seguido por el usuario *raíz* o un usuario *rama* y no superar los criterios de aceptación de usuarios que configuran el *Crawler*.
    - Ser seguido por un usuario *hoja* del conjunto de datos.
    - Haber publicado un *tweet* que ha sido *retwitteado* por un usuario final del conjunto de datos.
    - Haber sido mencionado en un *tweet* publicado por un usuario final del conjunto de datos.
    - Haber publicado un *tweet* que ha sido respondido por un usuario final del conjunto de datos.

  Estos usuarios pueden ser transformados en usuarios *para expandir* si son seguidos por un usuario *rama* o por el usuario *raíz* y superan los criterios de aceptación de usuarios que configuran el *Crawler* o pueden ser finalmente eliminados del conjunto de datos final por el programa *DbIsolator* si lo anterior no llega a suceder antes de la finalización del proceso de expansión de usuarios *rama*.

Estatus de un usuario en el conjunto de datos: Se diferencian cinco estatus posibles para cada usuario durante la construcción del conjunto de datos de los cuales sólo uno permanecerá en el conjunto de datos final tras la ejecución de los programas *Crawler* y *DbIsolator*:

- **Aceptado**: El usuario ha superado los criterios de aceptación de usuarios que configuran el *Crawler* y ha sido insertado en el mismo bien como usuario raíz en el comienzo de la construcción del conjunto de datos, bien como usuario *rama* u *hoja* como resultado de la expansión de un usuario *rama* o del usuario *raíz*.
- **Recolectado**: La información completa de usuario ha sido recuperada de un *tweet* publicado por el mismo y *retwitteado* por un usuario final del conjunto de datos cuyos *tweets* y *retweets* han sido solicitados a la API REST de *Twitter*. Aún no se ha comprobado si este usuario supera los criterios de aceptación de usuarios que configuran el *Crawler* pero su información no necesita ser solicitada de nuevo a la API en el

momento en el que se considere su inserción en el conjunto de datos por ser seguido por un usuario *rama* o por el usuario *raíz*.

- **Descartado**: El usuario no ha superado los criterios de aceptación de usuarios que configuran el *Crawler* y ha sido descartado como usuario final.
- **En proceso**: Los *tweets* más recientes de este usuario (que antes de tener este estado tenía el estado *Aceptado*) están siendo recuperados de la API REST de *Twitter*. Este estado intermedio permite la sincronización inter-instancia de distintas instancias del programa *Crawler* colaborando en la construcción de un mismo conjunto de datos.
- **Procesado**: Los tweets más recientes de este usuario ya han sido recuperados de la API REST de *Twitter*. Éste es el estado final de los usuarios definitivos del conjunto de datos tras la finalización de la ejecución del *Crawler*. Todos los usuarios cuyo estado final no sea *procesado* tras la finalización del proceso de *crawling* serán eliminados del conjunto de datos final por el programa *DbIsolator*.

En el apartado 2.1.1 del anexo 1 se puede observar cómo se gestiona a nivel de la base de datos asociada al conjunto de datos la información relativa a los distintos tipos de usuario, sus estados de expansión y sus status.

3.1.2 Ilustración gráfica del funcionamiento del programa *Crawler*

Vamos a ilustrar a continuación la expansión de un pequeño conjunto de datos de ejemplo asumiendo que el *Crawler* se configura para incluir un mínimo de 16 usuarios finales y que en cada expansión el número de nuevos usuarios que se eligen al azar como usuarios rama es 2.

En particular vamos a centrar nuestra representación en el proceso de expansión de usuarios ya que la recolección de *tweets* es un proceso simple en comparación con el de expansión y su representación gráfica complicaría los diagramas presentados.

Comenzamos la construcción del conjunto de datos con la inserción del usuario *raíz* u_0 tal y como se muestra en la figura 1.

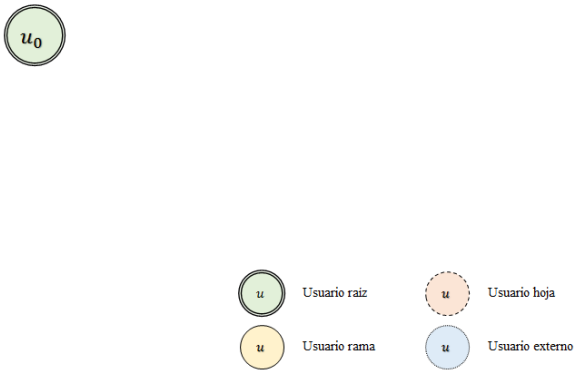


Figura 1. Comienzo de la construcción del conjunto de datos.

El primer paso para continuar con la construcción del conjunto de datos es expandir u_0: obtenemos de Twitter todos los usuarios a los que sigue u_0 y seleccionamos dos al azar como usuarios rama. Supongamos que se seleccionan los usuarios u_1 y u_6 como usuarios *rama* tal y como se muestra en la figura 2, estos usuarios servirán para la expansión posterior del conjunto de datos.

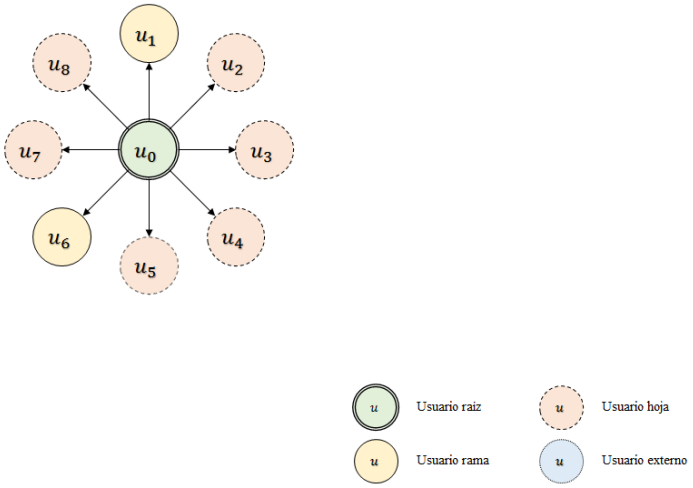


Figura 2. Expansión del usuario u_0, raíz del conjunto de datos.

El siguiente usuario que vamos a expandir, siguiendo la ordenación numérica de usuarios en esta simulación del funcionamiento del *Crawler*, es el usuario *rama* u_1 tal y como se muestra en la figura 3.

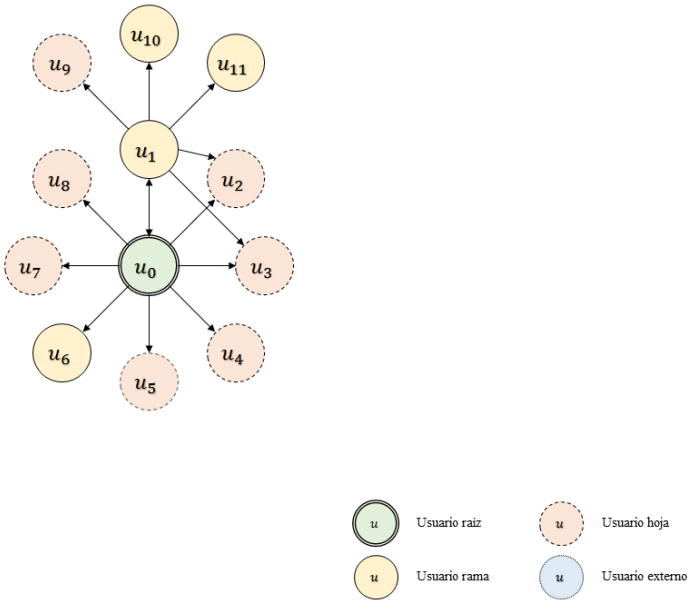


Figura 3. Expansión del usuario u_1, primer usuario rama del conjunto de datos.

El usuario *rama* u_1, sigue tanto a usuarios que ya estaban en el conjunto de datos en el momento de su expansión (u_0, u_2, u_3), cuyo tipo ya ha sido establecido en la expansión del usuario raíz u_1, como a usuarios que no estaban aún en el conjunto de datos en el

momento de su expansión (u_9, u_{10}, u_{11}) de los cuales u_{10} y u_{11} han sido insertados como nuevos usuarios *rama* y u_9 ha sido insertado como usuario *hoja*.

Proseguimos la construcción del conjunto de datos con la expansión de u_2, el primer usuario hoja tal y como se muestra en la figura 4.

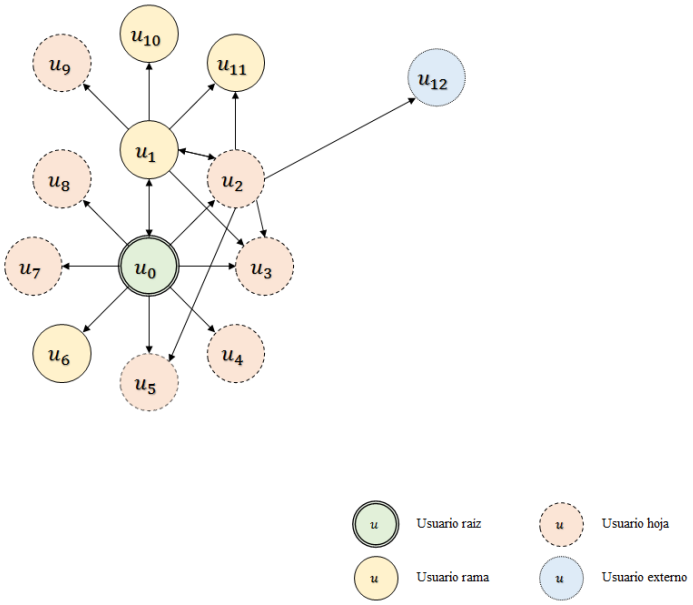


Figura 4. Expansión del usuario u_2, primer usuario hoja del conjunto de datos.

El usuario *hoja* u_2, sigue tanto a usuarios que ya estaban en el conjunto de datos en el momento de su expansión (u_1, u_3, u_{11}), como a un usuarios que no estaban aún en el conjunto de datos en el momento de su expansión (u_{12}). Dado que el usuario u_2 es un usuario *hoja*, el usuario u_{12} es considerado usuario externo del conjunto de datos: aún no es *rama* ni *hoja* pero la interacción *follow* dirigida del usuario u_2 al usuario u_{12} queda almacenada contemplando la posibilidad de que el usuario u_{12} sea seguido por alguno de los usuarios rama del conjunto de datos. En caso de que esto último no suceda, el usuario u_{12} así como la interacción *follow* dirigida de u_2 a u_{12} serán eliminados del conjunto de datos final por el programa *DbIsolator*.

Podemos continuar la construcción de nuestro conjunto de datos expandiendo los usuarios *hoja* u_3, u_4 y u_5 tal y como se muestra en la figura 5.

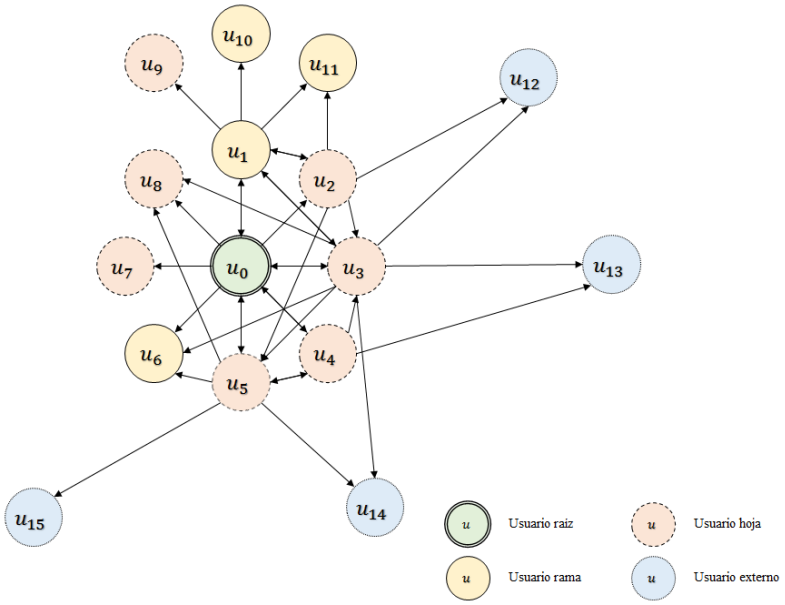


Figura 5. Expansión de los usuarios hoja u_3, u_4 y u_5.

La expansión de los usuarios *hoja* u_3, u_4 y u_5 ha añadido nuevos usuarios externos (u_{13}, u_{14} y u_{15}) al conjunto de datos así como nuevos links dirigidos de tipo *follow*. A continuación vamos a expandir el usuario rama u_6. Es importante observar el efecto que va a tener esta expansión sobre los usuarios externos u_{14} y u_{15} seguidos por los usuarios *hoja* u_3 y u_5. Al seguirlos el usuario *rama* u_6 van a pasar de ser usuarios externos a ser usuarios *hoja* o *rama* tal y como se muestra en la figura 6.

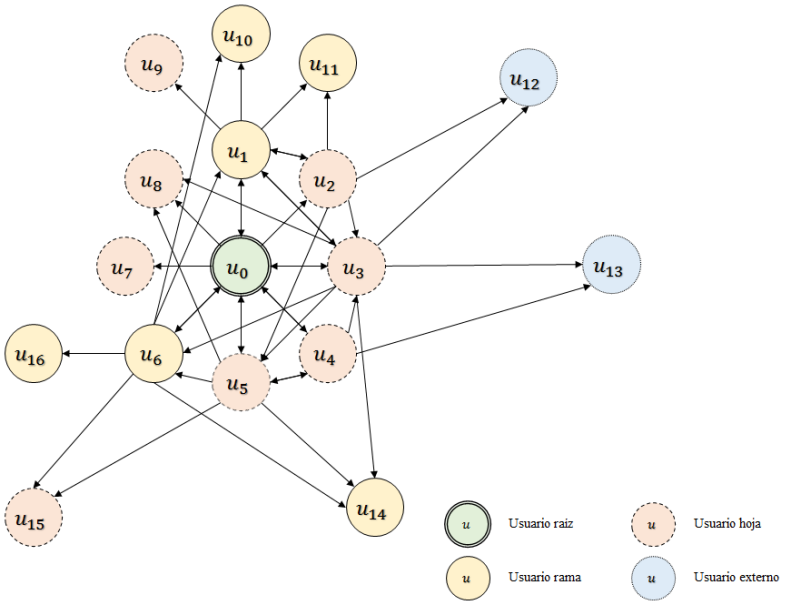


Figura 6. Expansión del usuario rama u_6.

El usuario rama u_6 sigue tanto a usuarios que ya estaban en el conjunto de datos en el momento de su expansión (u_0, u_1, u_{10}) como a usuarios que no estaban en el conjunto de datos en el momento de su expansión (u_{16}) o estaban como usuarios externos (u_{14}, u_{15}). En este caso el usuario u_{16} se incluye en el conjunto de datos como nuevo usuario *rama*, el usuario u_{14} pasa a ser también usuario *rama* y el usuario u_{15} pasa a ser usuario *hoja*.

Podemos continuar con la construcción de nuestro conjunto de datos expandiendo los usuarios hoja u_7, u_8 y u_9 tal y como se muestra en la figura 7.

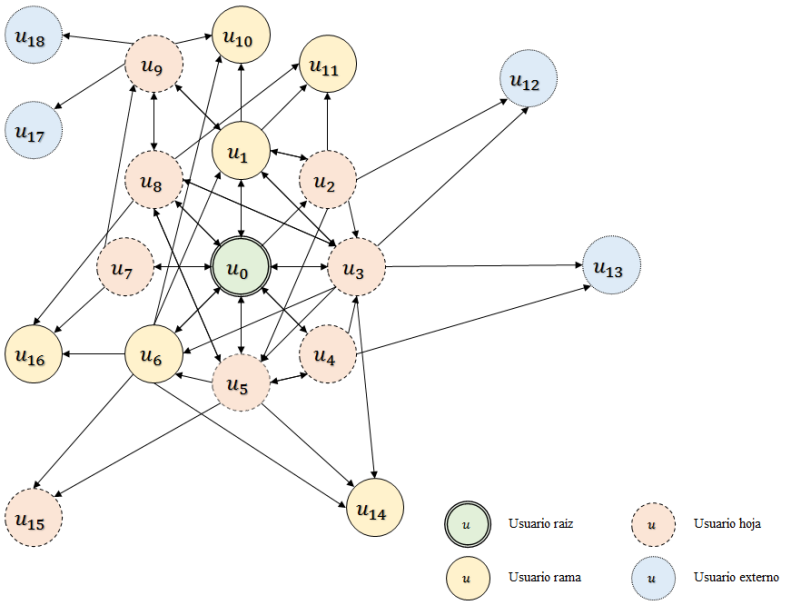


Figura 7. Expansión de los usuarios hoja u_7, u_8 y u_9.

De nuevo la expansión de usuarios *hoja* añade nuevos usuarios externos y enlaces *follow* al conjunto de datos. Procedemos a la expansión del usuario rama u_{10} tal y como se muestra en la figura 8.

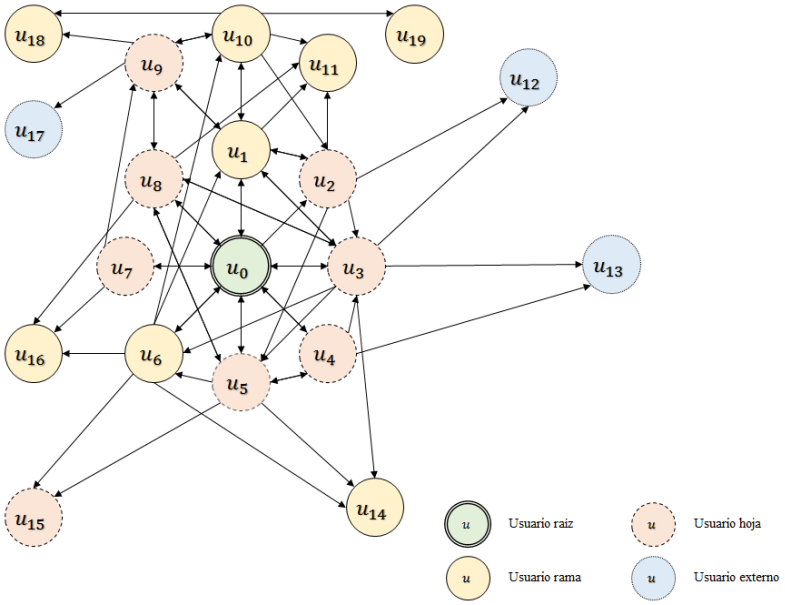


Figura 8. Expansión del usuario rama u_{10}.

Con la expansión del usuario u_{10} que conlleva la inserción en el conjunto de datos del nuevo usuario u_{19} y del usuario externo u_{18} como usuarios *rama*, conseguimos superar el número mínimo de 16 usuarios finales que nos proponíamos insertar en el conjunto de datos de ejemplo, a partir de este momento todo usuario *rama* que vaya a ser

expandido será transformado en usuario *hoja* antes de su expansión limitando así la extensión del conjunto de datos de ejemplo a los 17 usuarios finales que ya han sido insertados: u_0, u_1, u_2, u_3, u_4, u_5, u_6, u_7, u_8, u_9, u_{10}, u_{11}, u_{14}, u_{15}, u_{16}, u_{18} y u_{19}.

Fijémonos en la expansión del usuario u_{11} cómo es transformado en usuario *hoja* tal y como se muestra en la figura 9.

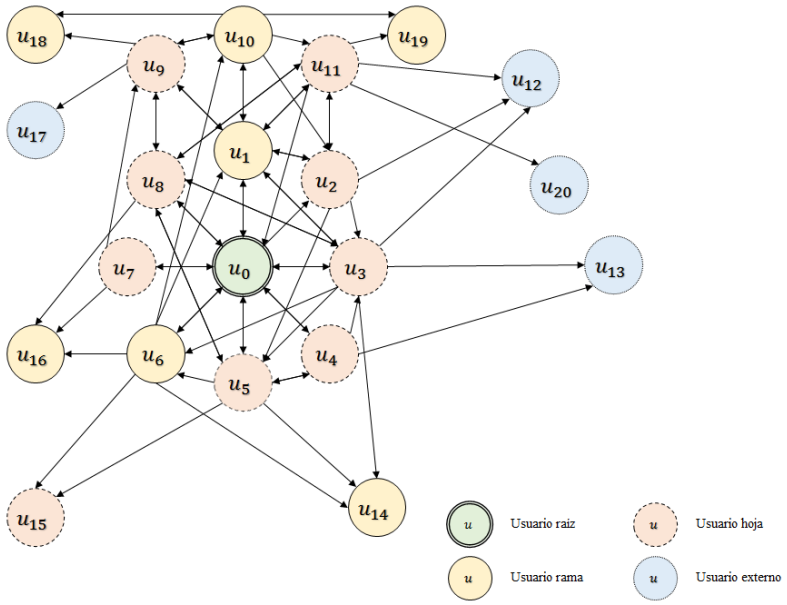


Figura 9. Expansión del usuario rama u_{11} que pasa a ser hoja.

Es importante observar, que, dado que el usuario rama u_{11} ha sido transformado en usuario hoja en el instante previo al comienzo de su expansión, a pesar de que siga al usuario externo u_{12} y al nuevo usuario externo u_{20}, éstos no van a ser incluidos como usuarios *hoja* o *rama* en el conjunto de datos sino que permanecerán a la espera de ser eliminados finalmente por el programa *DbIsolator*.

La expansión de los usuarios restantes del conjunto de datos (u_{14}, u_{15}, u_{16}, u_{18} y u_{19}) independientemente de si han sido insertados en el mismo como usuarios *rama* o como usuarios *hoja* se efectuará como expansión de usuarios *hoja* completando así el cierre del grafo de interacciones dirigidas *follow* entre los usuarios del conjunto de datos que va a contener tras las expansiones de estos usuarios todas las interacciones dirigidas *follow* observadas en *Twitter* entre los usuarios finales del conjunto de datos tal y como se muestra en la figura 10.

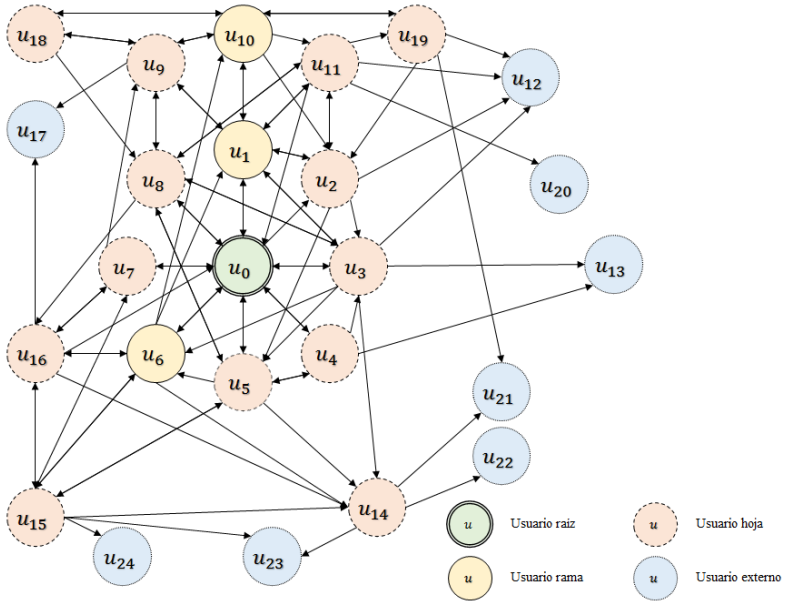


Figura 10. Expansión de los usuarios restantes del conjunto de datos u_{14}, u_{15}, u_{16}, u_{18} y u_{19}.

Dado que no queda ningún usuario más para ser expandido, concluye aquí la ejecución de nuestro *Crawler*. Como podemos observar, los usuarios u_{12}, u_{13}, u_{17}, u_{20}, u_{21}, u_{22}, u_{23} y u_{24}, han quedado fuera del conjunto de datos final. El programa *DbIsolator* va a ser el encargado de eliminar toda su información de la base de datos representativa del conjunto de datos final (no así del repositorio JSON ya que los datos recuperados pueden ser útiles para la construcción de nuevos conjuntos de datos).

3.1.3 Ilustración gráfica del funcionamiento del programa *DbIsolator*

El programa *DbIsolator* es el encargado de eliminar de la base de datos poblada por el programa *Crawler* toda la información correspondiente a usuarios que no forman parte definitivamente del conjunto de datos, es decir, aquellos usuarios cuya información ha sido recuperada a través de un *retweet* y no han llegado a ser seguidos por un usuario *rama* o por el usuario *raíz* o usuarios que son seguidos por al menos un usuario *hoja* y consecuentemente la arista del grafo de red social derivada de la acción *follow* ha sido recuperada por el programa *Crawler*.

Este proceso de aislamiento de la base de datos es necesario para dotar a la base de datos final de integridad referencial tal y como se describe detalladamente en el apartado 1.2 del anexo 1.

En lugar de eliminar los datos de la base de datos en la que ha intervenido el programa *Crawler*, el programa *DbIsolator* recupera los datos constituyentes del conjunto de datos final de la base de datos y los almacena en una nueva base de datos en la que se añaden las restricciones descritas el apartado 1.2 del anexo 1. Esto permite por una parte preservar los datos de *crawling* y por otra parte evitar la fragmentación en disco de la información almacenada en la base de datos final (ya que *DbIsolator* inserta en bloque la información final en la base de datos definitiva sin efectuar borrados en la misma).

Para concluir la ilustración gráfica de la construcción de un pequeño conjunto de datos ilustrativo iniciada en el apartado anterior, concluimos este apartado presentando el

efecto de la ejecución del programa *DbIsolator* seleccionando los datos finales (usuarios no externos) del conjunto de datos producido por el *Crawler* tal y como se muestra en la figura 11.

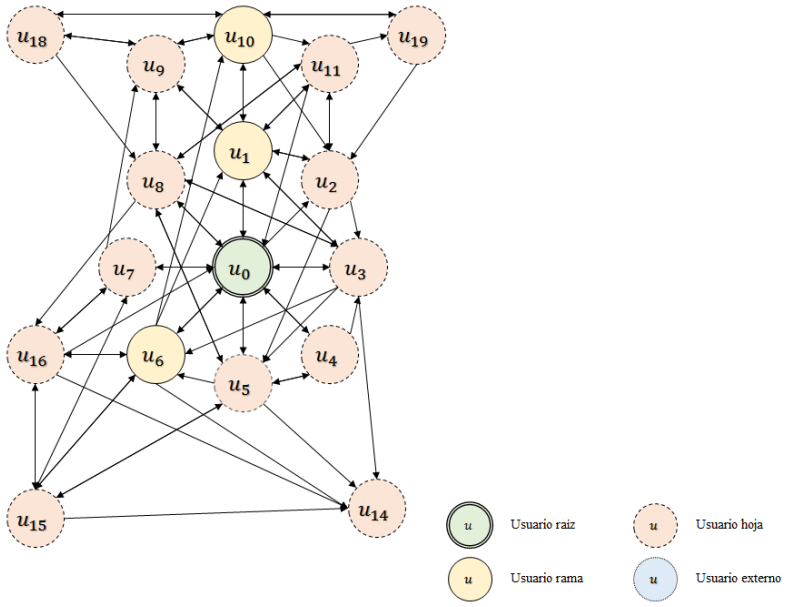


Figura 11. Actuación de DbIsolator sobre el conjunto de datos presentado en la figura 10.

De esta manera hemos obtenido un pequeño conjunto de datos ilustrativo con integridad referencial y englobando completamente el grafo de red social derivado de la acción *follow* dentro del subconjunto de *Twitter* que representan los 17 usuarios $\{u_0, u_1, u_2, u_3, u_4, u_5, u_6, u_7, u_8, u_9, u_{10}u_{11}, u_{14}, u_{15}, u_{16}, u_{18}, u_{19}\}$.

3.2 Detalles técnicos y decisiones de diseño

Una vez nos hemos familiarizado con los programas *Crawler* y *DbIsolator* podemos entrar en aspectos técnicos tales como los distintos parámetros de configuración que admite el programa *Crawler*, los detalles relativos a su interacción con la API REST de *Twitter* o su arquitectura multihilo y multi instancia.

3.2.1 Configuración del Crawler

El programa *Crawler* acepta tres tipos distintos de argumentos de configuración que permiten determinar las dimensiones del conjunto de datos así como establecer criterios de aceptación de usuarios en el mismo y por último configurar la ejecución del mismo de cara a un mayor aprovechamiento de los recursos computacionales de la máquina en la que se ejecute:

- **Aceptación de usuarios**: Estos argumentos establecen los requisitos que deben cumplir los distintos usuarios de *Twitter* para ser aceptados como usuarios finales en el conjunto de datos construido. En particular permiten establecer el número mínimo de tweets publicados por los mismos, el número mínimo y máximo de usuarios seguidos y el tiempo mínimo transcurrido desde la creación de un usuario hasta la solicitud de su información de usuario a la API REST de *Twitter*.

- **Descubrimiento de usuarios y tamaño final del conjunto de datos**: Estos argumentos establecen las líneas de funcionamiento del *Crawler* permiten definir cuál es el primer usuario insertado en el conjunto de datos, cuántos usuarios debe contener el conjunto de datos final como mínimo, cuántos tweets se recuperarán como máximo de cada usuario y si el recorrido de *Twitter* recuperando usuarios se realiza en anchura, en profundidad o siguiendo una estrategia híbrida. Esto último se decide indicando el número máximo de usuarios que se escogen al azar como usuarios raíz entre los usuarios seguidos por un usuario raíz o por el usuario rama: para hacer un recorrido en profundidad se debe indicar que se escoja un único usuario tras cada expansión, para hacer un recorrido en anchura se debe indicar que se escojan todos los usuarios tras cada expansión y para seguir una estrategia se debe especificar el número máximo de usuarios que se escojan.
- **Rendimiento**: Se ofrece un argumento de configuración para establecer el número de hilos de ejecución que serán lanzados para recuperar la información de todos los usuarios de cada lista de usuarios seguidos por un usuario empleado para ampliar el conjunto de datos incluyendo los usuarios que éste sigue en el conjunto de datos.

En el anexo 3 se incluye una descripción completa, específica y exhaustiva del funcionamiento de los argumentos de ejecución de este programa. En el apartado 2.3 del anexo 2 se detalla además cómo distintas instancias del programa *Crawler* que colaboren en la preparación de un mismo conjunto de datos sincronizan su configuración.

3.2.2 Recuperación de información de *Twitter*

Para interactuar con la API REST de *Twitter* se hace uso de la biblioteca java *Twitter4J*[1]. En particular toda la información que necesitamos para completar el conjunto de datos se obtiene con los métodos `getFriendsIDs`, `showUser` y `getUserTimeline` de la interface `Twitter` de dicha biblioteca que hacen uso respectivamente de las peticiones a la API REST de *Twitter* `GET friends/ids`, `GET users/show` y `GET statuses/user_timeline`. A continuación resumimos estas tres peticiones. La documentación completa de las mismas, junto con el resto del API, está disponible en la fuente original de *Twitter*: https://dev.twitter.com/docs/api.

La petición `GET friends/ids` devuelve la lista de identificadores de usuario de los usuarios que sigue un usuario de *Twitter* y consecuentemente es la que vamos a utilizar en el proceso de *expansión* de un usuario para recuperar los usuarios que éste usuario sigue e introducir en el conjunto de datos las interacciones *follow* de este usuario con los mismos y, en caso de que el usuario sea *raíz* o *rama*, incluir los nuevos usuarios en el conjunto de datos siempre y cuando superen los criterios de aceptación que configuran el *Crawler*.

El código JSON que devuelve la API REST de *Twitter* en respuesta a esta petición se almacena en el fichero comprimido en zip `Follow/[id_usuario].zip` dentro del

[1] Ver http://twitter4j.org/en/index.html

repositorio JSON asociado al conjunto de datos de forma que antes de realizar una petición `GET friends/ids` se pueda comprobar si ya se dispone de esta información. Esto último es muy importante para la eficiencia del programa *Crawler* ya que esta petición tiene una limitación de uso de 15 peticiones cada 15 minutos.

La petición `GET users/show` devuelve información descriptiva variada de un usuario. En particular devuelve toda la información necesaria para determinar si el usuario supera los criterios de aceptación de usuarios que configuran el *Crawler*.y toda la información de usuario que se almacena en la tabla *User*. Consecuentemente esta llamada es la que vamos a utilizar en el proceso de *expansión* de usuarios para recuperar la información de los usuarios seguidos por un usuario *rama* o por el usuario *raíz* que no formen parte aún del conjunto de datos y consecuentemente sean susceptibles de ser incluidos en el mismo.

El código JSON que devuelve la API REST de *Twitter* en respuesta a esta petición se almacena en el fichero comprimido en zip `User/[id_usuario].zip` dentro del repositorio JSON asociado al conjunto de datos de forma que antes de realizar una petición `GET users/show` se pueda comprobar si ya se dispone de esta información. Al igual que sucedía con la llamada anterior, almacenar esta información en JSON y acceder a ella para evitar repeticiones de llamadas a la API REST de *Twitter* es muy importante para la eficiencia del programa *Crawler* ya que esta petición tiene una limitación de uso de 180 peticiones cada 15 minutos.

La petición `GET statuses/user_timeline` devuelve los *tweets* más recientes publicados por un usuario siempre y cuando su configuración de privacidad permita el acceso público de los mismos. Esta información nos permite para cada *tweet* recuperado de cada usuario final del conjunto de datos completar una entrada en la tabla *Tweet* así como extraer la información relativa a todas las interacciones *mention*, *reply* y *retweet* del usuario cuyos *tweets* recuperamos. Consecuentemente esta llamada es la que vamos a utilizar en el proceso de recuperación de los *tweets* más recientes de los usuarios finales del conjunto de datos.

El código JSON que devuelve la API REST de *Twitter* en respuesta a esta petición se divide en tantos tweets como hayan sido recuperados en función de la configuración del *Crawler* y la actividad del usuario hasta el momento de la petición. El código JSON de cada uno de estos *tweets* se almacena en el fichero comprimido en zip `Status/[id_tweet].zip` dentro del repositorio JSON asociado al conjunto de datos de forma que la información contenida en el conjunto de datos relativa a cada *tweet* pueda ser extendida en un futuro. Además, en el caso particular de los *retweets*, si la información del usuario autor de este tweet no ha sido solicitada aún a la API y consecuentemente almacenada en el repositorio JSON, se recupera el JSON que describe a este usuario del propio *retweet* y se almacena este código en el fichero comprimido en zip `User/[id_usuario].zip` dentro del repositorio JSON. Al igual que sucedía con la llamada anterior, *Twitter* limita su uso a 180 peticiones cada 15 minutos.

3.2.3 Arquitectura multi-hilo y multi-instancia del programa *Crawler*

Como hemos podido observar en el sub apartado anterior, existen restricciones severas en el uso de la API REST de *Twitter* que hacen que un único *Crawler* trabajando de forma secuencial tenga muy poca capacidad de recuperación de información. En particular la limitación de uso de la petición `GET friends/ids` restringe considerablemente la capacidad de realización del resto de peticiones en una arquitectura *mono-hilo*.

Esto ha motivado dos decisiones de diseño en la arquitectura del *Crawler*: por una parte un diseño *multi-hilo* del *Crawler* que evite que la espera provocada por la superación del límite de uso de uno de los tipos de peticiones impida hacer uso de los otros tipos de peticiones en una misma instancia del *Crawler* y por otra parte un diseño *multi-instancia* que permita a distintas instancias del *Crawler* colaborar juntas en la recuperación de información para generar el conjunto de datos, haciendo uso de distintas cuentas de *Twitter* con sus respectivas tasas de utilización de la API.

En lo que respecta al diseño multi-hilo del *Crawler* se utilizan tres tipos distintos de hilo, cada uno asociado a uno de los tres pares método de *Twitter4J* - petición a la API REST de *Twitter* presentados con anterioridad.

Una instancia del *Crawler* despliega inicialmente un hilo `UsersRetrievalThread` y un hilo `TweetsRetrievalThread`. El primero de ellos se encarga de expandir y recuperar usuarios hasta que se alcanza el número mínimo de usuarios establecido en la configuración del *Crawler* y se recuperan todos los link *follow* que parten de los usuarios finales del conjunto de datos. El segundo de ellos se encarga de recuperar los tweets más recientes publicados por los usuarios que han sido aceptados como usuarios finales del conjunto de datos cuyos *tweets* más recientes aún no han sido recuperados.

El hilo `UsersRetrievalThread` a su vez despliega tantos hilos `FolloweesRetriever` como se indique en la configuración de rendimiento del *Crawler*. Estos hilos son los encargados de recuperar la información asociada a los usuarios seguidos por el usuario *raíz* o por un usuario *rama* en proceso de expansión. La figura 12 resume la arquitectura multi-hilo del programa *Crawler*. En este diagrama la caja rectangular que ocupa la posición superior representa el hilo principal de ejecución del programa *Crawler* y las cajas con bordes suavizados representan los distintos hilos asociados a los pares método – llamada. Las líneas que unen los distintos hilos representan las relaciones hilo padre – hilo hijo en tiempo de ejecución del programa *Crawler* siendo el hilo padre el que se sitúa en el extremo superior y el hilo hijo el que se sitúa en el extremo inferior.

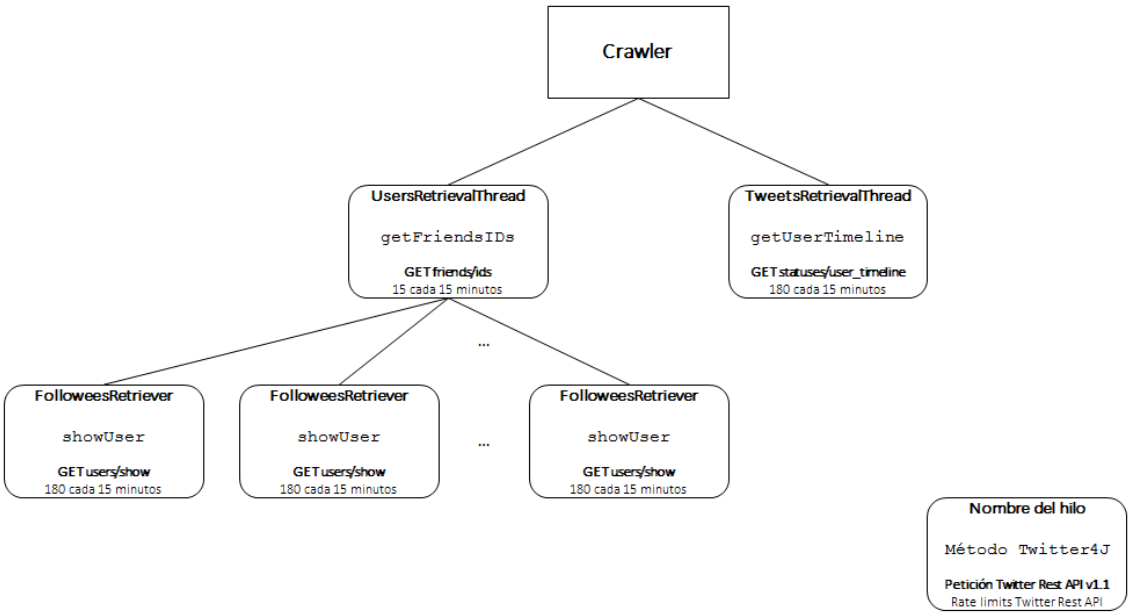


Figura 12. Diagrama de la arquitectura multi-hilo del Crawler.

En lo que respecta a la sincronización *multi-instancia* se ha optado por usar la base de datos como elemento de coordinación ya que las posibilidades de configuración del nivel de aislamiento transaccional del motor de bases de datos MySQL permite controlar la lectura y escritura de datos de forma segura. En particular se han configurado todas las interacciones con la base de datos MySQL como *prepared statements* que se han integrado en transacciones cuyo nivel de asilamiento ha sido configurado como SERIALIZABLE[2].

Por una parte, se ha incluido en una tabla de configuración la información de configuración que permite que todas las instancias de *Crawler* que colaboren en la construcción de un mismo conjunto de datos sigan el mismo procedimiento de expansión y apliquen los mismos criterios de aceptación de usuarios. Por otra parte en la tabla que recoge la información de los usuarios del conjunto de datos se ha incluido la información de coordinación que permite a cada instancia del *Crawler* determinar la información que ya ha sido recuperada, aquella que está siendo recuperada y la que no ha sido recuperada aún. Se puede consultar la estructura de esta información de forma detallada en el apartado 2.1.1 del anexo 2.

Estas dos decisiones de diseño han permitido en poco tiempo generar el conjunto de datos escenario de los experimentos que hemos realizado con un total de 10.029 usuarios y más de 2.000.000 *tweets*.

[2] Ver http://dev.mysql.com/doc/refman/5.6/en/set-transaction.html#isolevel_serializable.

3.3 Estructura del conjunto de datos

Llegados a este punto estamos preparados para describir formalmente la estructura del conjunto de datos preparado que, recapitulando, consiste en un conjunto de usuarios de *Twitter*, *tweets* escritos por los usuarios, conexiones *follow* entre los usuarios, y una serie de datos adicionales relativos a usuarios y *tweets*. Esta información se ha almacenado como base de datos SQL empleando MySQL como gestor de bases de datos. Además se acompaña esta base de datos de información complementaria almacenada en un repositorio JSON que recoge toda la información provista por la API REST de *Twitter* en cada petición HTTP GET realizada por el programa *Crawler* descrito con anterioridad.

El diagrama Entidad Relación de la base de datos diseñada se muestra en la figura 13.

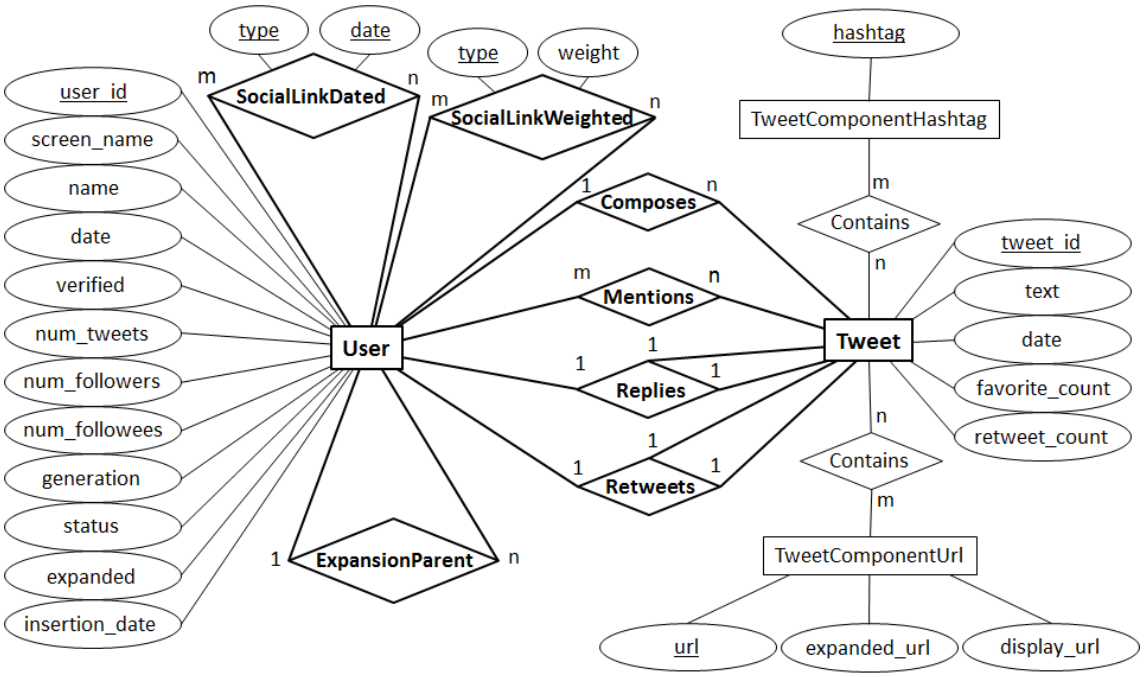


Figura 13. Diagrama ER del conjunto de datos.

Las dos tablas principales en este conjunto de datos son *User* y *Tweet*. En ellas se representan respectivamente los usuarios (sujetos candidatos para recomendación de otros usuarios o tweets con los que interactuar) y los *tweets* (información contextual asociada a los usuarios).

El resto de las tablas representan componentes informativos de los *tweets* y de los usuarios así como relaciones entre los mismos. Adicionalmente la base de datos contiene la tabla *Settings* cuyo diagrama ER es el que se muestra en la figura 14.

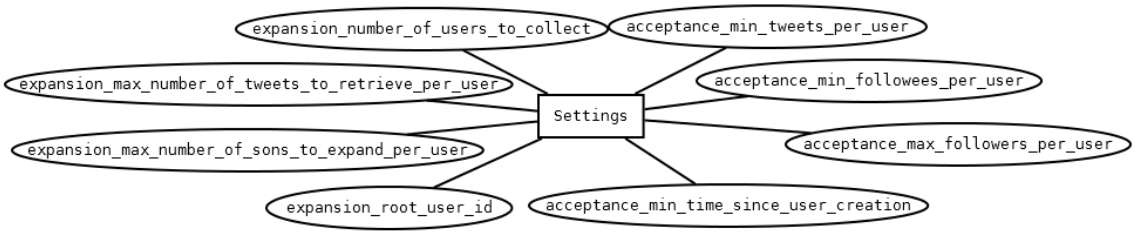


Figura 14. Diagrama ER de la Tabla Settings.

Se trata de una tabla con fines implementales en la construcción del conjunto de datos al igual que parte de los campos de la tabla de usuarios tal y como se indicó en el apartado 3.2.3.

En el anexo 1 se presenta el código SQL que describe la base de datos completa incluyendo las restricciones que garantizan integridad referencial en la misma y en el anexo 2 se describe en detalle el contenido y la utilidad de los campos de todas las tablas y relaciones presentadas en los diagramas ER de la base de datos.

Esta estructura de base de datos ha sido elegida para adecuarse a la taxonomía propuesta en (Kywe, et al., 2012). No obstante, esta estructura no recoge toda la información de usuarios y *tweets* que ofrece *Twitter* a través de su API REST. Como ya se ha indicado previamente, este es uno de los motivos que ha motivado a complementar la base de datos con un repositorio JSON extendido que almacena toda la información ofrecida por la API en cada petición de usuarios, *tweets* o usuarios seguidos: por una parte almacena repositorio de la información adicional ofrecida por *Twitter* y no almacenada en la base de datos y por otra minimiza el número de llamadas a la API REST de *Twitter* durante la construcción del conjunto de datos.

3.4 Análisis del conjunto de datos

Para construir el conjunto de datos escenario de los distintos experimentos de recomendación realizados, se han empleado los programas *Crawler* y *DbIsolator*, con los parámetros de configuración del programa *Crawler* que se detallan en el apartado 2.3 del anexo 2. Como resultado de esta ejecución hemos obtenido un conjunto de datos compuesto por un total de 10.029 usuarios y 2.028.097 *tweets*. En estos datos se pueden considerar diferentes estructuras de red social dirigida.

Por un lado la red natural sería la derivada de la relación *follow*. Dada la metodología de construcción del conjunto de datos, el grafo resultante se compondrá necesariamente de una única componente débilmente conexa. Los usuarios rama y el usuario raíz presentan un *out-degree* especialmente superior al del resto de usuarios dado el sesgo hacia este *out-degree* que motiva el algoritmo de expansión del programa *Crawler*. No obstante, este último detalle no es excesivamente influyente dada la distribución final de los usuarios del conjunto de datos que se muestra en la figura 15.

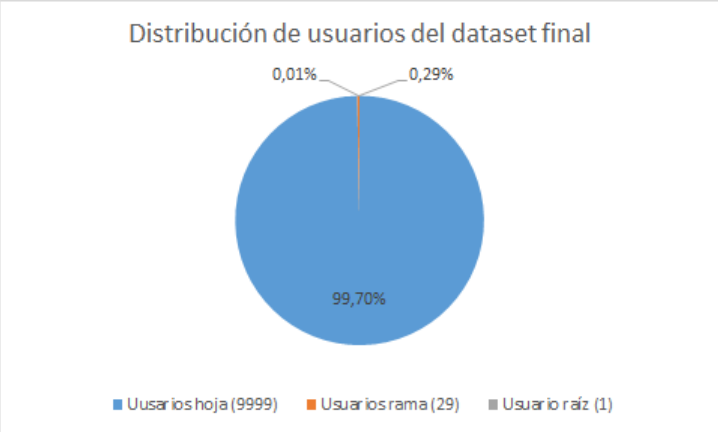


Figura 15. Distribución de los usuarios finales en las categorías hoja, rama y raíz.

No se dispone de información temporal asociada a las aristas de este grafo debido a las especificaciones de la API REST de *Twitter*. Por este motivo, a la hora de separar el conjunto de datos en un conjunto de datos de entrenamiento y un conjunto de datos de test para experimentar con recomendaciones que empleen algoritmos basados en red social, la forma más razonable de dividir este grafo consiste en realizar una partición aleatoria.

Además del grafo explícito de seguidores, es posible considerar el grafo implícito inducido por la interacción entre usuarios a través de sus tweets. Puede ser tanto o más significativo que un usuario *retwittee* varios posts de otro, aun cuando aquél no siga directamente a éste, que el hecho de que en algún momento un usuario añada a otro entre sus seguidos, pero en la práctica no muestre interés por sus tweets interactuando con ellos. Así pues en este trabajo consideramos la red social implícita derivada de las acciones *mention*, *reply* y *retweet* observadas en los *tweets* del conjunto de datos. Este grafo dispone de más información asociada que el grafo anterior: tiempo de establecimiento de las aristas y tipo de arista. Además dada su relación directa con los *tweets* con los cuales comparte información temporal, se trata de un grafo propicio a la separación en conjunto de entrenamiento y test mediante partición temporal para la experimentación de algoritmos de recomendación basados en red social y en información contextual, es decir, la contenida en los tweets de entrenamiento. En este caso ya no observamos una única componente débilmente conexa (aunque sí una típica componente gigante y unos pocos usuarios aislados apartados de la misma) y el sesgo de *out-degree* hacia los usuarios rama y el usuario raíz queda mucho más atenuado que en el grafo anterior.

Consideramos pues dos grafos con distinta naturaleza y posibilidades de aprovechamiento que nos van a permitir comparar resultados experimentales en uno y en otro. Para dar una idea de las características y topología de estas dos redes, mostramos en la tabla 1 algunas métricas y estadísticas obtenidas con la herramienta *Gephi*[3]:

[3] Ver https://gephi.org/

	Follow	Mention / Reply / Retweet
Número de nodos	10029	9874
Número de aristas	560227	137241
Grado saliente promedio	55.8605	13.649
Densidad	0.006	0.001
Componentes débilmente conexas	1	81
ASL	3.710	5.494
Diámetro	11	18
Coeficiente de clustering promedio	0.193	0.095
Modularidad	0.526	0.572

Tabla 1. Métricas calculadas con Gephi en los dos grafos de red social.

Lo primero que observamos es la diferencia entre el número de nodos del grafo *Follow* y el grafo *Mention / Reply / Retweet*. (al cual nos vamos a referir en lo sucesivo como grafo de interacción) Hay un total de 155 usuarios presentes en el primer grafo que no lo están en el segundo, es decir, usuarios que a pesar de participar en la red social siguiendo a usuarios del conjunto de datos, no han registrado interacciones con ningún usuario del conjunto de datos en sus *tweets*.

Si comparamos el número de aristas vemos que en el grafo *Follow* hay del orden de 4 veces el número de aristas que en el grafo de interacción, es decir, a pesar de tener un vecindario amplio en el grafo de *Follow*, los usuarios no han interactuado recientemente con todos sus vecinos. Llama la atención no obstante observar que la cantidad de interacciones registradas con fecha en la base de datos es de 553284, es decir, la redundancia promedio de las aristas del grafo de interacción es del orden de 4. Tomando en consideración el dato presentado al inicio de este párrafo y el último dato recuperado podemos concluir que curiosamente los usuarios del conjunto de datos tienden a interactuar reiteradamente con aproximadamente el 25% de su vecindario.

Observamos que, como es natural en redes sociales, el nivel de densidad global es muy bajo. Más llamativa es la diferencia entre el número de componentes débilmente conexas entre los dos grafos. El grafo *Follow*, naturalmente y dado el procedimiento de construcción del conjunto de datos descrito anteriormente cuenta con una única componente débilmente conexa. Por contrapartida el grafo de interacción presenta un total de 81 componentes débilmente conexas. Este número tan elevado de componentes nos motiva a proceder a un análisis más detallado del grafo con la herramienta *Gephi* para comprobar que la composición en componentes débilmente conexas del grafo de interacción es la que se puede observar en el siguiente diagrama:

El grafo de interacción se distribuye en una componente débilmente conexa gigante de 9792 usuarios (99,1% de los usuarios del grafo), dos componentes de dos usuarios cada una y 79 componentes de un usuario cada una (usuarios que se mencionan o contestan a sí mismos y a nadie más en los *tweets* recogidos en el conjunto de datos). En este sentido, dado el pequeño porcentaje que representan los usuarios que no pertenecen a la

componente conexa principal y el pequeño tamaño de las componentes externas a la misma, la diferencia en el número de componentes débilmente conexas entre los dos grafos no constituye una gran diferencia entre ambos.

Una diferencia más marcada entre los dos grafos sí que la encontramos midiendo las longitudes de los caminos más cortos entre todos los pares de usuarios de los dos grafos: La métrica ASL representa la longitud promedio en número de aristas de los caminos más cortos entre cada par de nodos y, como podemos observar es considerablemente más baja en el grafo *Follow* (3.710) que en el grafo de interacción (5.494). En concordancia con esta métrica observamos la diferencia entre el Diámetro de los dos grafos, que representa la longitud del camino más largo entre todos los caminos más cortos entre pares de nodos conectados en cada grafo: En el caso del grafo *Follow* (11) es más baja que en el caso del grafo de interacción (18). Si tenemos en cuenta la diferencia de densidad entre los dos grafos comentada anteriormente, es natural encontrar distancias más cortas en el grafo *Follow* que en el grafo de interacción. Esto, como observaremos en apartados posteriores, va a favorecer el funcionamiento de algoritmos centrados en el vecindario cercano en el caso del grafo *Follow* y va a favorecer resultados de novedad y diversidad en el grafo de interacción.

En lo que respecta al coeficiente de clustering, observamos cierta proporcionalidad con la densidad del sub-grafo representativo de los vecindarios de cada usuario. El grafo *Follow* presenta un coeficiente superior al del grafo de interacción (0.193 frente a 0.095).

Cabe destacar por último la simulitud de los dos grafos cuando comparamos su modularidad (0.526 en el grafo *Follow* y 0.572 en el grafo de interacción), éste valor lo obtiene *Gephi* asignando una clase de modularidad a cada nodo y reagrupando los nodos iterativamente para maximizar la asortatividad de clase del grafo siguiendo el algoritmo propuesto en (Blondel 2008).

Podemos visualizar en la figura 16 sendos grafos coloreando los nodos en función de la clase de modularidad que maximiza este valor, determinando su tamaño en función de su *in-degree* y posicionándolos en el plano mediante un algoritmo automático *force atlas 2* de la herramienta *Gephi*[4].

[4] Ver https://gephi.org/2011/forceatlas2-the-new-version-of-our-home-brew-layout/

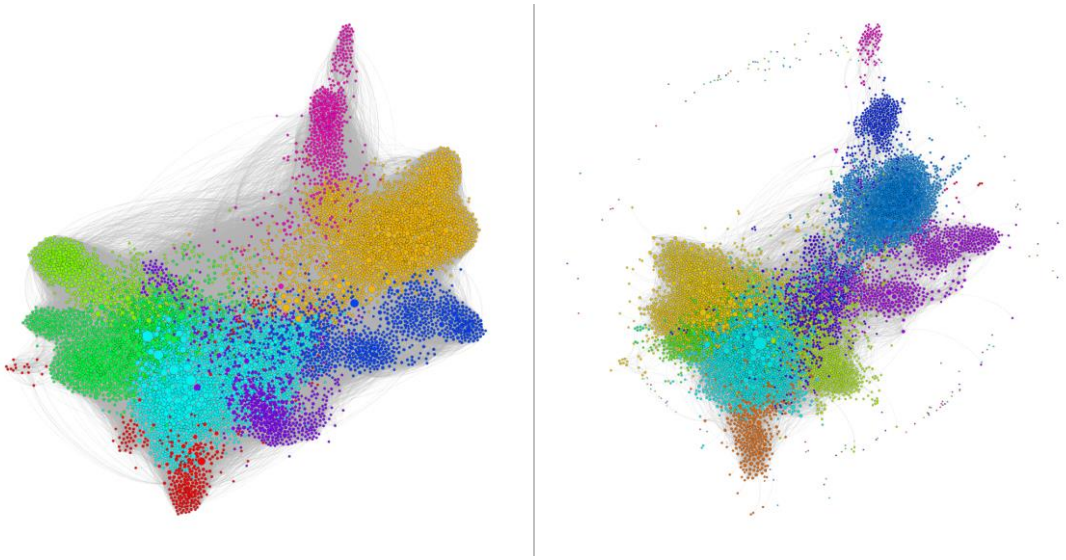


Figura 16. Visualización de los grafos Follow (izquierda) y de interacción (derecha).

Como podemos observar, se aprecia la diferencia en la densidad del conjunto de aristas de los dos grafos. Se aprecia también en el grafo de la derecha las componentes débilmente conexas en los márgenes de la figura, cosa que no se da en la figura de la izquierda donde todos los nodos pertenecen a la misma componente.

Si nos fijamos en el aspecto de las componentes y en el coloreado de los módulos, encontramos a ojos vista cierta correlación entre las clases que maximizan modularidad en el grafo de la izquierda y las que la maximizan en el grafo de la derecha. Esto es natural si tenemos en cuenta que, dada la construcción de sendos grafos, la mayor parte de las interacciones contenidas en el grafo de la derecha se daban ya en el grafo de la izquierda.

Para concluir nuestro análisis del conjunto de datos vamos a analizar la distribución del grado de todos los nodos de cada uno de los dos grafos. En particular vamos a visualizar en la figura 17 el *out-degree*, el *in-degree* y el *grado* de cada uno de los usuarios de los dos grafos contando el número de usuarios por grado y presentando la información en diagramas de dispersión utilizando la escala logarítmica tanto en el eje horizontal como en el vertical de forma que se puedan apreciar mejorlas tendencias *power law* –en caso de haberlas-en las distribuciones.

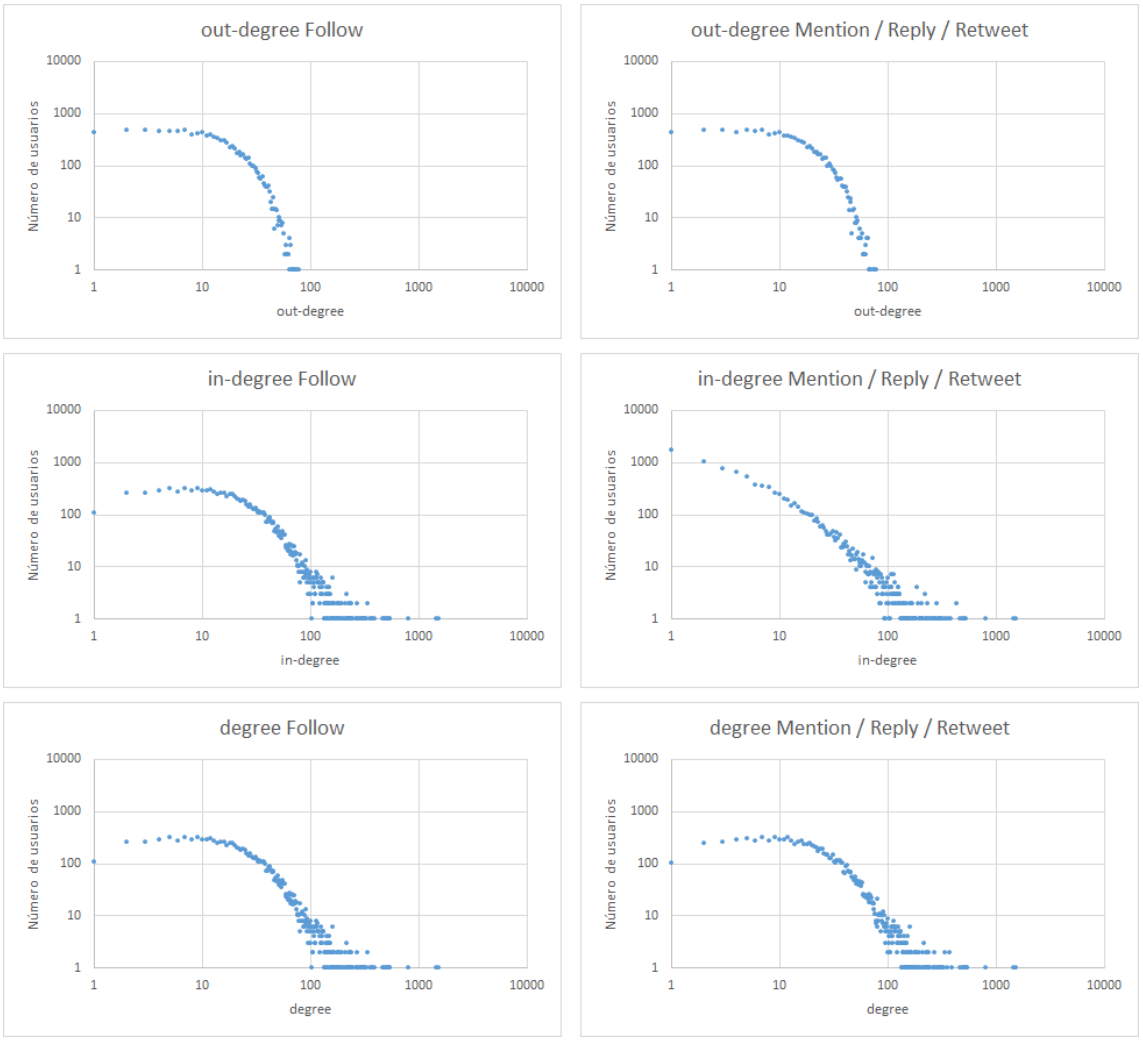


Figura 17. Distribución del grado.

Como podemos observar, los dos grafos presentan tendencias muy similares en lo que respecta a la distribución del grado de sus respectivos nodos. A partir de grados del orden de 20 se observa una clara tendencia power law (es decir una forma de tendencia rectilínea en la escala log-log) en todas las gráficas. También encontramos un sesgo en grados bajos debido a la selección de usuarios mínimamente activos y conectados entre sí durante la construcción del conjunto de datos.

Destaca especialmente la gráfica de *in-degree* del grafo de interacción donde prácticamente no se observa el sesgo anteriormente mencionado. Esto se debe a que la influencia del sesgo es mayor en el grafo *Follow*, que es más dependiente del algoritmo de expansión del conjunto de datos y es muy posible que se encuentre relacionado con el hecho de que el *in-degree* está íntimamente relacionado con el nivel de popularidad de los usuarios dentro del conjunto de datos, lo cual puede motivar la tendencia de power law.

Es importante tener en cuenta que en el grafo de interacción encontramos tres tipos distintos de interacción. En la figura 18 se divide la columna derecha de la figura presentada sobre estas líneas para mostrar el grado por separado de el grafo correspondiente a cada uno de los tres tipos de interacción.

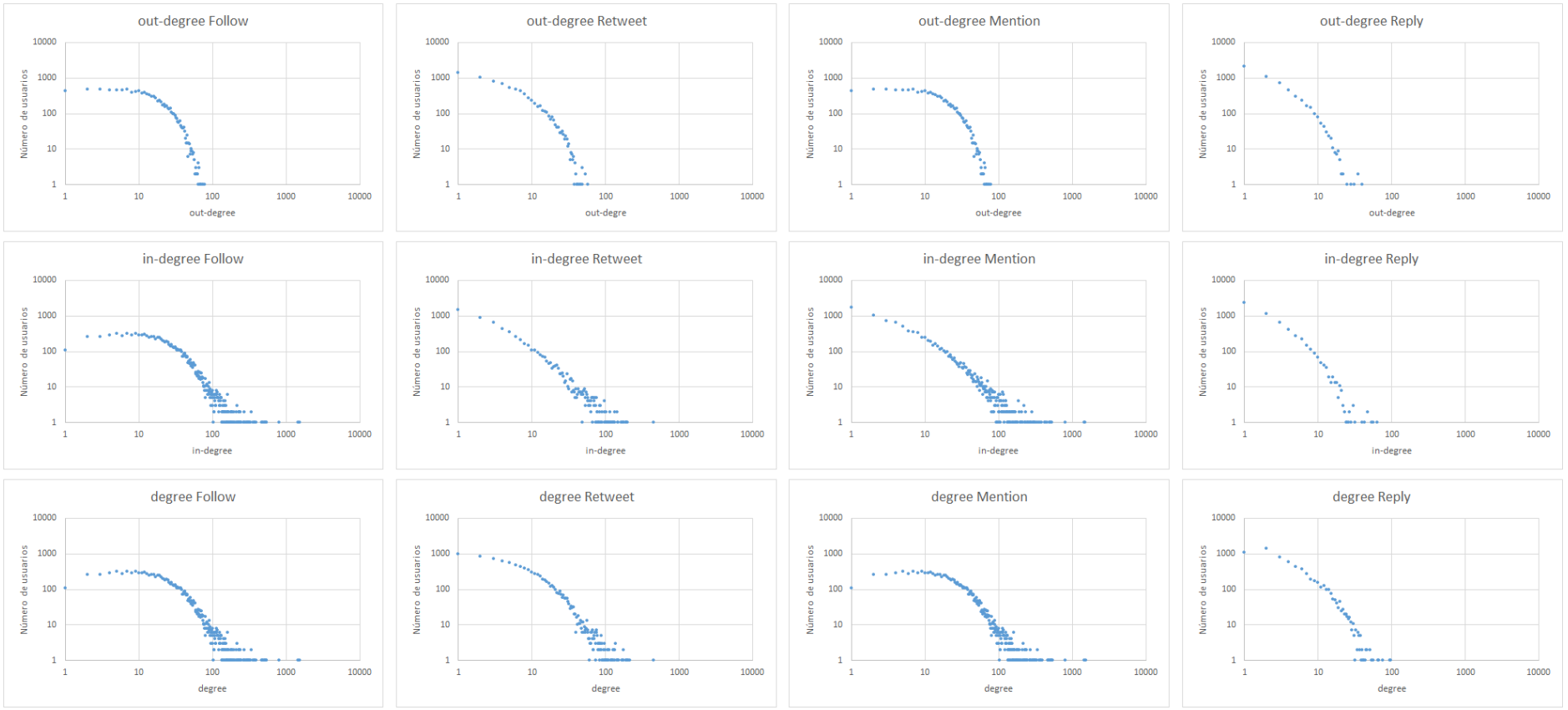


Figura 18. Distribución del grado por tipos de interacción.

Como podemos observar, la distribución del grado en los grafos *Retweet* y *Mention* es realmente similar. En lo que respecta a densidad observamos que el grafo *Reply* es el menos denso de los tres mientras que el grafo *Mention* es el más denso. Esto se entiende si tenemos en cuenta que cada interacción *Retweet* y *Reply* conlleva una interacción mention y que la interacción *Retweet* es la más popular en *Twitter* dada su naturaleza como red social de difusión de información.

4. Plataforma de experimentación

Para la realización sistemática de experimentos de recomendación sobre el conjunto de datos descrito en el capítulo anterior, hemos desarrollado una plataforma de experimentación que permita integrar distintos algoritmos de recomendación y métricas de evaluación de forma rápida y sencilla implementando una interfaz (de recomendación o evaluación) y declarando un nuevo recomendador o evaluador en el sistema. Dentro de esta plataforma englobamos el flujo de datos completo desde la red social *Twitter* hasta los ficheros finales de evaluaciones y automatizamos la generación de ficheros de recomendación y ficheros de evaluación haciendo uso de los distintos algoritmos declarados.

4.1 Estructura de la plataforma

La plataforma se compone de un total de 10 programas ejecutables cuyas entradas y salidas de datos están relacionadas en secuencia. La plataforma incluye un programa adicional de ayuda para obtener información acerca del funcionamiento y la configuración por defecto de la plataforma de experimentación, cuya salida se muestra en el anexo 3.

Los dos primeros programas de los que se compone la plataforma son los presentados en el apartado 3.1 *Crawler* y *DbIsolator*, encargados de la construcción del conjunto de datos generando la base de datos *MySQL* que almacena la red social y la información de los tweets, junto con el repositorio JSON. Procesan la información contenida en esta base de datos dos nuevos programas: *Graphgen* y *Dataset2Zip* que transforman la información contenida en la base de datos para permitir la visualización de los grafos dirigidos de red social presentes en el conjunto de datos y el funcionamiento optimizado de algoritmos de recomendación. La salida del programa *Dataset2Zip*, que representa el conjunto de datos de experimentación, es separada en dos subconjuntos de entrenamiento y test haciendo uso del programa *Splitter* para permitir realizar experimentos de recomendación y evaluar los mismos mediante distintas métricas. Los programas *Indexer*, *InteractionMapBuilder* y *CentroidBuilder* son los encargados de procesar el texto contenido en los Tweets publicados por cada usuario para generar representaciones vectoriales de los mismos que permitan realizar recomendaciones basadas en contenido. Por último los programas *RecomendationsGenerator* y *Evaluator* generan recomendaciones y las evalúan, respectivamente, finalizando así el flujo de datos de la plataforma de experimentación. En la figura 19 se representa gráficamente este flujo de datos así como la estructura de la plataforma derivada del mismo.

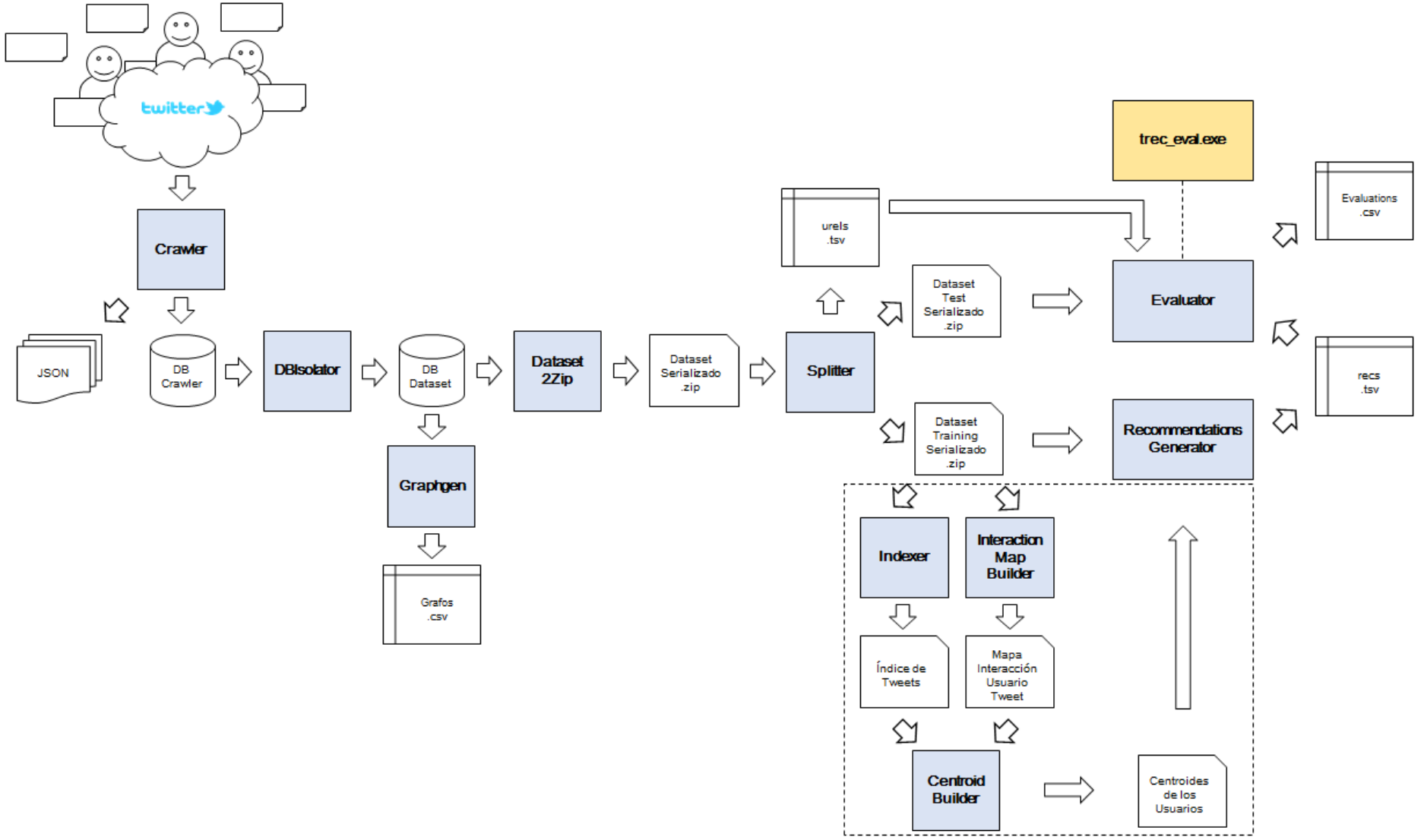


Figura 19. Plataforma de experimentación para recomendación en Twitter.

A continuación procedemos a describir detalladamente la funcionalidad y características más destacables de cada uno de los programas componentes de la plataforma de recomendación así como el formato de los datos que reciben como entrada y los datos que producen completando así la explicación del diagrama presentado en la figura 19.

4.2 Crawler

El programa *Crawler*, presentado en el apartado 3.1, constituye el punto de partida de la plataforma de experimentación. Su papel dentro de la plataforma de experimentación es el de obtener toda la información de *Twitter* necesaria para generar el conjunto de datos y su salida es por una parte el repositorio JSON con toda la información provista por la API REST de *Twitter* y por otra parte una base de datos con toda la información del conjunto de datos escenario desprovista de integridad referencial completa por las razones que se presentan en el apartado anterior.

4.3 DbIsolator

El programa *DbIsolator*, presentado en el apartado 3.1, finaliza la construcción del conjunto de datos iniciada por el programa *Crawler* recuperando de la base de datos generada por el *Crawler* la información definitiva constitutiva del conjunto de datos escenario final de experimentación y trasladándola a la base de datos final, que sí tiene integridad referencial completa.

Tras la ejecución de estos dos primeros programas, la plataforma de experimentación cuenta con un conjunto de datos escenario completo y estructurado almacenado en una base de datos *MySQL* e información complementaria en formato JSON almacenada en disco y comprimida en zip que hace posible, en caso de necesidad, la extensión de la base de datos para incluir más información asociada a *tweets* y usuarios (ubicaciones geográficas, idiomas asociados a los usuarios, etc) sin necesidad de recurrir a la API REST de *Twitter* para recuperar de nuevo la información.

4.4 Graphgen

El programa *Graphgen* tiene como objetivo hacer compatible con la herramienta de tratamiento de grafos *Gephi* la información de red social contenida en la base de datos generada por los dos programas anteriores. Para ello extrae de la base de datos producida por los dos programas anteriores la información relativa a los grafos dirigidos de interacción *Follow*, *Mention*, *Reply* y *Retweet* y genera distintos ficheros separados por comas con formato compatible con el asistente de importación de *Gephi*[5] que permiten analizar y visualizar los grafos de red social contenidos en el conjunto de datos generado.

[5] Ver https://wiki.gephi.org/index.php/Import_CSV_Data

4.5 Dataset2Zip

El programa *Dataset2Zip* transforma el conjunto de datos almacenado en la base de datos en un objeto java y serializa el mismo para almacenarlo finalmente comprimido en zip.

Dada la naturaleza distinta de los grafos de red social presentes en la base de datos (el grafo *Follow* por una parte sin información temporal asociada y el grafo de interacción, separable en distintos sub-grafos, con información temporal y contextual asociada en forma de *tweets*) a partir de la misma base de datos este programa genera distintos tipos de conjunto de datos serializado (en función del tipo o tipos de interacción o interacciones con los cuales se configure el programa para su ejecución).

En particular se diferencian dos representaciones del conjunto de datos como objeto Java dependientes del tipo o tipos de interacción considerada en su construcción: una primera representación básica limitada prácticamente al grafo de red social sin información proveniente de *tweets* para el caso de interacciones de tipo *follow* que denominamos `TwitterData` y su extensión `TimeAwareTwitterData` que mantiene junto al grafo de red social los *tweets* cuyas interacciones recoge el grafo. En la figura 20 se presenta el diagrama UML de estas dos clases.

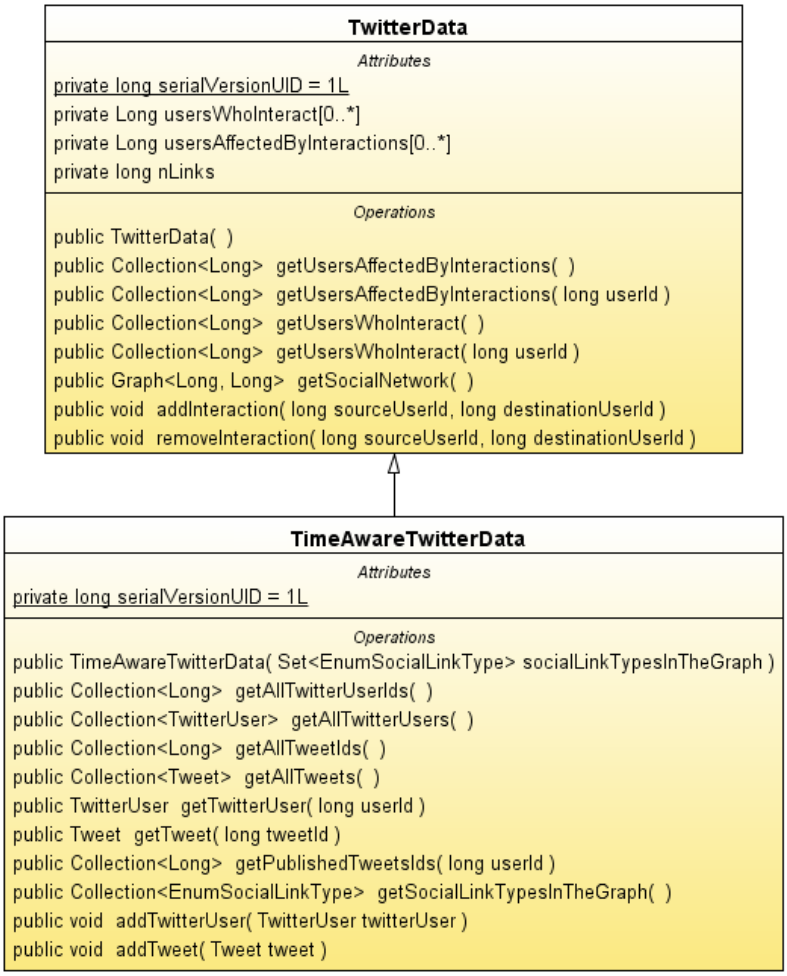


***Figura 20. Diagrama UML de* `TwitterData` *y* `TimeAwareTwitterData`.**

- `TwitterData`: Es la clase base de representación del conjunto de datos, asociada a interacciones de tipo *follow*. Dado que las interacciones de tipo *follow* no tienen información temporal asociada y los *tweets* sí la tienen, esta clase limita su representación de la base de datos a un grafo dirigido de interacciones modelado como una instancia de `DirectedSparseGraph` de la biblioteca Java JUNG (*Java Universal Network/Graph Framework*)[6] acompañado de una colección de representaciones como objeto Java de los usuarios presentes en el conjunto de datos. Esto evita incoherencias a la hora de separar los datos en conjuntos de entrenamiento y test para experimentar con recomendaciones de usuarios ya que, en caso de mantener conjuntamente la red social por *follow* sin información temporal y la información asociada a los *tweets*, podrían darse casos en que los propios *tweets* desvelasen interacciones sociales por *follow* (por ejemplo en el caso de dos usuarios que interactuasen frecuentemente en el conjunto de datos de entrenamiento pero las relaciones *follow* que los relacionan hubiesen quedado en test).
- `TimeAwareTwitterData`: Esta clase extiende a la clase `TwitterData` y está asociada a las interacciones de tipo *mention*, *reply* y *retweet*. Esta vez el grafo JUNG representa las interacciones del tipo o tipos indicados en la configuración de ejecución del programa (en el anexo 3 se pueden observar los parámetros de configuración comunes y específicos de cada programa de la plataforma) y se amplía la información serializada con una colección de representaciones como objetos Java de los *tweets* publicados por los distintos usuarios del conjunto de datos.

Es importante tener en cuenta que en nuestros experimentos no vamos a manejar simultáneamente relaciones de tipo *Follow* y relaciones inducidas por las interacciones *Mention*, *Reply* o *Retweet*, por motivos que explicaremos en el apartado 4.6.

La serialización de los datos y su almacenamiento comprimido en memoria permiten a los próximos programas de la plataforma de experimentación cargar en memoria el subconjunto completo del conjunto de datos que van a emplear para la realización de experimentos, agilizando considerablemente la producción de resultados experimentales. No obstante, en un escenario de mayor escala, donde el tamaño del conjunto de datos impidiese su carga completa en memoria, sería necesario gestionar el acceso a los datos en memoria física y el almacenamiento parcial de los datos en memoria durante el proceso de recomendación. Pero no se da este caso en los experimentos que aquí se documentan.

4.6 Splitter

El programa *Splitter* es el encargado de particionar el conjunto de datos serializado generado por la herramienta *Dataset2Zip* en dos subconjuntos de entrenamiento y test que permitan por una parte generar recomendaciones y por otra parte evaluar las recomendaciones generadas. Además genera el fichero de juicios de relevancia representativo de las interacciones entre usuarios contenida en el conjunto de datos de entrenamiento en un

[6] Ver http://jung.sourceforge.net/

formato compatible con el formato del input de la herramienta de evaluación *trec_eval*[7]. Esta herramienta se utiliza en las campañas de evaluación TREC, de referencia en el campo de la Recuperación de Información. trec_eval representa la implementación de referencia de una amplísima gama de métricas incluyendo las clásicas y otras más recientes.

La metodología de separación del conjunto de datos en entrenamiento y test varía en función del tipo de `TwitterData` tomado como input para la realización de la partición:

- **Partición aleatoria**: Para separar conjuntos de datos de tipo `TwitterData`, es decir, conjuntos de datos sin información temporal ni tweets asociados, se procede separando el grafo de interacciones sociales en dos grafos disjuntos seleccionando de forma aleatoria un porcentaje de las aristas (especificado como parámetro de ejecución del *Splitter*) para el conjunto de entrenamiento y el resto de las aristas para el conjunto de test. De este modo se generan dos `TwitterData` que contienen la información de todos los usuarios del conjunto de datos y los dos subconjuntos del grafo original y un fichero de juicios de relevancia representativo del conjunto de datos de test.
- **Partición cronológica**: Para separar conjuntos de datos de tipo `TimeAwareTwitterData`, es decir, conjuntos de datos con tweets y consecuentemente información temporal asociada, se procede ordenando los tweets cronológicamente, generando el `TimeAwareTwitterData` a partir de los primeros *tweets* del conjunto de datos original (tomando el porcentaje de tweets especificado como parámetro de ejecución del *Splitter*) y utilizando el resto de los *tweets* para generar el `TwitterData` de test y su fichero de juicios de relevancia correspondiente.
  Es importante tener en cuenta que hay interacciones en los últimos *tweets* que pueden haberse dado previamente en los *tweets* empleados para generar el `TimeAwareTwitterData` de entrenamiento. Es decir, un usuario u puede haber *retwitteado* dos *tweets* de otro usuario v, uno antes y otro después del punto temporal de corte de la partición. Cuando esto es así, recomendar a u que siga a v daría lugar a un acierto trivial. Para evitar estos solapamientos entre el grafo de entrenamiento y el grafo de test, las interacciones ya aparecidas en el grafo de entrenamiento se excluyen del grafo de test durante su generación. De este modo se garantiza que el grafo de entrenamiento y el grafo de test, al igual que en el caso de la partición aleatoria, conformen una partición del grafo original.

De cara a experimentos futuros se ha optado por diseñar de forma flexible el conjunto de datos permitiendo que se puedan integrar y seleccionar mediante argumentos de ejecución distintos algoritmos de separación del conjunto de datos.

[7] Ver https://wiki.umiacs.umd.edu/adapt/index.php/Webarc:Trec_Eval_(modified)

4.7 Indexer

El programa *Indexer* es el primer programa del mecanismo de procesamiento offline de los *tweets* asociados a los conjuntos de datos de entrenamiento de tipo `TimeAwareTwitterData` junto con los programas *InteractionsMapBuilder* y *CentroidBuilder*. El objetivo conjunto de estos tres programas es generar representaciones vectoriales de los usuarios del conjuntos de datos de entrenamiento en función del contenido textual de los *tweets* que han publicado para permitir generar recomendaciones basadas en contenido.

En particular el programa *Indexer* emplea la biblioteca Java de indexación *Apache Lucene*[8] para generar un índice tratando cada *tweet* del `TimeAwareTwitterData` como un documento de texto. Dado que *Lucene* asigna identificadores secuenciales a cada documento indexado por motivos de eficiencia, el programa *Indexer* complementa este índice con un mapa serializado que relaciona los identificadores de los *tweets* del `TimeAwareTwitterData` indexado con los identificadores internos que Lucene les asigna en su índice.

Es importante tener en cuenta que este programa, al igual que los programas *InteractionsMapBuilder* y *CentroidBuilder,* sólo es compatible con conjuntos de datos de entrenamiento de tipo `TimeAwareTwitterData` ya que estos son los que contienen tweets que puedan ser indexados y su ejecución sólo es necesaria para generar recomendaciones basadas en contenido, por este motivo este conjunto de tres programas se ha destacado en el diagrama de la figura 19 como componente complementario del flujo de datos de la plataforma de experimentación.

4.8 InteractionsMapBuilder

El programa *InteractionsMapBuilder* es el segundo programa del mecanismo de procesamiento offline de *tweets* asociados a los conjuntos de datos de entrenamiento de tipo `TimeAwareTwitterData`. Su objetivo es generar un mapa serializado y comprimido en zip que relacione cada usuario con el conjunto de *tweets* que va a ser tomado para la generación de su representación vectorial asignando un peso de relevancia a cada *tweet* dentro de esta representación.

Siguiendo la filosofía flexible de la herramienta de experimentación se ha diseñado este programa de forma que soporte la definición, declaración y uso seleccionable mediante parámetros de ejecución de distintos algoritmos de generación del mapa de interacciones. En el capítulo de algoritmos de recomendación procedemos a describir en detalle los algoritmos de generación del mapa de interacciones que hemos implementado y utilizado en nuestros experimentos.

[8] Ver http://lucene.apache.org/core/

4.9 CentroidBuilder

El programa *CentroidBuilder* es el tercer y último programa del mecanismo de procesamiento offline de la información contextual en forma de *tweets* asociada a los conjunto de datos de entrenamiento de tipo `TimeAwareTwitterData`. A partir del índice generado por el programa *Indexer* y el mapa de interacciones generado por el programa *InteractionsMapBuilder* genera los *centroides* representativos de la información contextual asociada a cada usuario del conjunto de datos.Estos *centroides* consisten en una representación vectorial de cada usuario en la cual las coordenadas se corresponden con los términos aparecidos en los *tweets* tomados para la generación de los *centroides*. Los pesos de cada término presente en el *centroide* se calculan como el promedio ponderado por peso asignado a cada par *tweet*-usuario por el programa *InteractionsMapBuilder*, es decir el peso del término en la representación vectorial *tf-idf* de cada *tweet* tal y como se muestra en la figura 22.

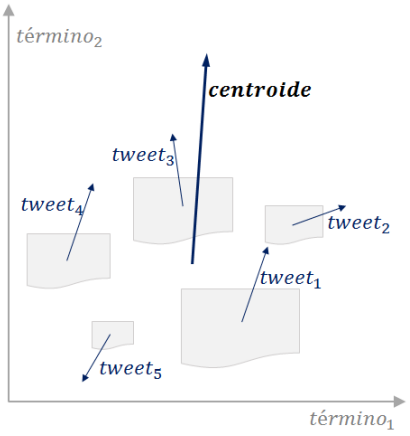


Figura 21. Representación conceptual de la representación vectorial de un usuario.

La representación vectorial *tf-idf* de cada *tweet* toma como coordenadas los términos aparecidos en el *tweet* y como pesos asociadas a las mismas la ponderación *tf-idf* del término asociado a la coordenada:

$$tf - idf(t,d) = tf(t,d) \cdot idf(t) \quad (1)$$

Donde t representa un término y d representa un documento (*tweet*).

- $tf(t,d)$ mide la "importancia" del término t en el documento (*tweet*) d. Típicamente se define mediante una función creciente respecto a la frecuencia de t en d.
- $idf(t)$ mide el poder de discriminación del término t. . Es habitual definir idf mediante una función decreciente respecto a la frecuencia de t en la colección, como medida de la especificidad de t.

En la figura 22 se muestra una representación gráfica simplificada del sistema de representación vectorial de tweets *tf-idf*.

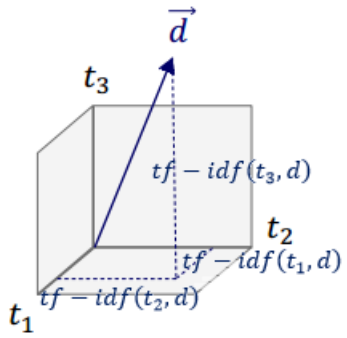


Figura 22. Representación conceptual de la representación vectorial de un tweet por tf-idf.

En nuestra implementación hemos optado por la implementación típica de las funciones tf e idf:

$$tf(t,d) = \begin{cases} 1 + \log_2 frec(t,d) \text{ si } frec(t,d) > 0 \\ 0 \text{ si } frec(t,d) = 0 \end{cases} \quad (2)$$

$$idf(t) = \log\left(\frac{|\mathfrak{D}|}{|\mathfrak{D}_t|}\right) \quad (3)$$

Donde $frec(t,d)$ es el número de apariciones del término t en el tweet d, $\mathfrak{D}$ es la colección de *tweets* completa del `TimeAwareTwitterData` indexado y $\mathfrak{D}_t$ es la sub-colección de *tweets* del `TimeAwareTwitterData` en los que aparece el término t.

4.10 RecommendationsGenerator

El programa *RecommendationsGenerator* constituye el núcleo de la plataforma de pruebas: es el encargado de generar recomendaciones a partir del conjunto de datos de entrenamiento. Este programa se ha diseñado e implementado con el objetivo de permitir la máxima flexibilidad posible: mediante argumentos de ejecución es posible decidir los algoritmos de recomendación que se quieren ejecutar y los parámetros de configuración de los mismos en caso de que el algoritmo permitiese su configuración. Para más información acerca de los parámetros de configuración del programa *RecommendationsGenerator* se se remite al lector al anexo 3.

A nivel implemental, los algoritmos de recomendación se abstraen a la implementación de una interfaz común de recomendación de forma que para implementar y probar un nuevo algoritmo de recomendación sólo es necesario implementar dicha interfaz y declarar correctamente el algoritmo dentro de un tipo enumerado para permitir al intérprete de argumentos de ejecución ubicar el algoritmo correspondiente de forma que pueda ser empleado en la generación de recomendaciones.

En particular nos hemos centrado en la recomendación de usuarios, no obstante, la interfaz de recomendación así como el conjunto de datos `TimeAwareTwitterData` son completamente compatibles con la recomendación de tweets ya que se consideran como posibilidades para iterar recomendaciones tanto el conjunto de usuarios como el conjunto

de *tweets* y en función de dónde se declare el algoritmo, el programa *RecommendationsGenerator* iterará sobre usuarios o sobre *tweets*.

De cara a optimizar el rendimiento a la hora de generar recomendaciones, el programa *RecommendationsGenerator* admite como parámetro de configuración el número de hilos de ejecución que se desea mantener durante la generación de recomendaciones. En función de este parámetro, el programa distribuye equitativamente los usuarios susceptibles de recibir recomendaciones en subconjuntos de usuarios y genera un hilo de recomendación para cada subconjunto. Esto exige que las implementaciones de la interfaz de recomendación sean *thread-safe*.

El resultado de la ejecución de este programa, para cada algoritmo de recomendación y cada configuración del mismo, es un fichero de recomendaciones que recoge los scores de recomendación y los identificadores de los pares usuario-ítem recomendado en un formato compatible con el formato del input de la herramienta de evaluación *trec_eval*[9].

En el capítulo 5 procedemos a la descripción detallada de los algoritmos de recomendación implementados y probados haciendo uso de la plataforma de pruebas y en particular del programa *RecommendationsGenerator*.

4.11 Evaluator

El programa *Evaluator* cierra el flujo de datos de la plataforma de experimentación. Su objetivo es evaluar las recomendaciones generadas por el programa *RecommendationsGenerator* utilizando por una parte los ficheros de recomendaciones generados por dicho programa y por otra los juicios de relevancia y el `TwitterData` de test producidos por el programa *Splitter*.

Al igual que en el caso del programa *RecommendationsGenerator*, este programa se ha diseñado e implementado con el objetivo de permitir la máxima flexibilidad posible: mediante argumentos de ejecución es posible decidir los algoritmos de evaluación que se quieren ejecutar y los parámetros de configuración de los mismos. De nuevo remitimos al anexo 3 para más información acerca de los parámetros de configuración del programa.

A nivel implemental, los algoritmos de evaluación se abstraen a la implementación de una interfaz común de evaluación de forma que, para implementar y probar un nuevo algoritmo de evaluación sólo es necesario implementar dicha interfaz y declarar correctamente el algoritmo dentro de un tipo enumerado para permitir al intérprete de argumentos de ejecución ubicar el algoritmo correspondiente de forma que pueda ser empleado en la evaluación de recomendaciones.

Además de los algoritmos de evaluación que se pueden desarrollar implementando la interfaz común de evaluación, el programa *Evaluator* interactúa con el programa externo *trec_eval* que permite calcular distintas métricas de precisión a partir de los ficheros

[9] Ver https://wiki.umiacs.umd.edu/adapt/index.php/Webarc:Trec_Eval_(modified)

de recomendaciones y juicios de relevancia producidos por los programas *RecommendationsGenerator* y *Splitter* respectivamente.

El resultado de la ejecución de este programa es un fichero de evaluaciones representativo de las evaluaciones de los distintos algoritmos de recomendación, en formato csv (separado por comas) orientado al análisis de los resultados de las evaluaciones con Excel. En este fichero cada fila representa un experimento (la evaluación mediante un algoritmo de evaluación con una configuración específica de un fichero de recomendaciones producido por un algoritmo de recomendación con una configuración específica) con las siguientes 9 columnas:

- **Urels file path**: Path en el que se ubica el fichero de juicios de relevancia empleado por el programa *Evaluator* en caso de que el experimento haya hecho uso del fichero de juicios de relevancia, `NO_URELS_FILE_USED` en caso de que el algoritmo de recomendación no haya hecho uso del fichero de juicios de relevancia obteniendo el input necesario del `TwitterData` de test.
- **Recs file path**: Path en el que se ubica el fichero de recomendaciones que se han evaluado en el experimento. El programa *RecommendationsGenerator* sigue un protocolo de nomenclatura de ficheros de recomendaciones que permite obtener de este path la información relativa al algoritmo de recomendación empleado y su configuración así como los tipos de interacciones sociales considerados en el conjunto de datos de experimentación.
- **Recommendation algorithm**: Información relativa al algoritmo de recomendación evaluado y sus parámetros de configuración.
- **Metric**: Información relativa a la métrica de evaluación empleada y sus parámetros de configuración.
- **Recs cutoff**: Número de ítems (en nuestro caso usuarios) recomendados a cada usuario como máximo en el fichero de recomendaciones evaluado.
- **Threshold**: Valor mínimo de relevancia asociado a un ítem susceptible de ser recomendado en el fichero de juicios de relevancia para ser considerado un acierto. En el caso particular de los ficheros de juicios de relevancia generados por el programa *Splitter* sólo se consideran dos posibles valores de relevancia:
  - 1 si en un subconjunto de test existe una arista dirigida desde el usuario susceptible de recibir recomendaciones y el usuario recomendado.
  - 0 en caso contrario.

  En un escenario de recomendación de tweets tendríamos igualmente dos posibles valores:

  - 1 si el usuario ha manifestado interés por el *tweet* (ya sera mediante un *retweet* o un *reply*) en el subconjunto de test.
  - 0 en caso contrario.

  Por este motivo, en esta columna sólo vamos a encontrar dos valores posibles: 1 si se usa el fichero de juicios de relevancia en el experimento, `NO THRESHOLD` en caso contrario.

- **All urels values used**: Verdadero si se utilizan todos los valores del fichero de juicios de relevancia a la hora de generar la evaluación, falso si se utiliza el fichero de juicios de relevancia a la hora de generar la evaluación pero no se utiliza la totalidad de la información contenida en el mismo y `NO URELS FILE USED` en caso de que se prescinda del fichero de juicios de relevancia en el experimento y la información se obtenga del `TwitterData` de test.
- **User id**: Identificador del usuario empleado en el experimento. También puede aparecer la palabra clave `all` si el experimento promedia –como es habitual– la métrica sobre todos los usuarios del conjunto de datos. En el segundo caso, el programa *Evaluator* admite en sus parámetros de configuración la opción de decidir si agregar las evaluaciones promediando sobre todos los usuarios o generar una fila del fichero de evaluaciones para cada usuario.
- **Evaluation**: Score final de evaluación generado por la métrica o algoritmo de evaluación como conclusión del experimento.

En el capítulo 6 procederemos a la descripción detallada de los algoritmos de evaluación empleados y analizamos los resultados obtenidos tras evaluar los distintos algoritmos de recomendación con los que hemos experimentado.

5. Algoritmos de recomendación

En la presente sección presentamos los algoritmos de recomendación que se han implementado y probado en el conjunto de datos escenario presentado con anterioridad.

Estos algoritmos asignan a cada usuario susceptible de ser recomendado un valor numérico (al que nos referiremos como *score*) que será empleado para comparar este usuario con el resto de los usuarios a la hora de generar una lista ordenada de recomendaciones. El conjunto final de recomendaciones que se generan para un usuario es la lista de usuarios con los que no se ha observado interacción en el conjunto de datos de entrenamiento ordenada en orden descendente por *score*.

Todos los algoritmos han sido implementados orientándose a la recomendación de usuarios a los que seguir. Estos algoritmos se dividen en dos grupos en función de la naturaleza de los datos de entrenamiento y test que emplean. Por una parte se han implementado algoritmos basados exclusivamente en el grafo dirigido de red social que representa las interacciones entre los usuarios y por otra parte, algoritmos basados en el contenido de los *tweets* publicados por los usuarios. A partir de este punto, vamos a referirnos al primer tipo de algoritmos como **algoritmos basados en la topología de red social** y al segundo tipo como **algoritmos basados en el contenido**.

Además de todos los algoritmos de recomendación que se exponen bajo estas líneas, se ha implementado un **sistema de hibridación de algoritmos** que linealiza la distribución de los *socores* generados por los distintos algoritmos y permite combinar algoritmos a la hora de generar recomendaciones. Esto último se explicará al final de la presente sección y será utilizado en los distintos experimentos sobre el conjunto de datos escenario.

5.1 Algoritmos basados en la topología de red social

Estos algoritmos explotan distintas características medibles del grafo de red social a la hora de realizar recomendaciones.

Antes de proceder a presentar los algoritmos basados en la topología de red social, vamos a establecer la notación que vamos a emplear a la hora de referirnos a los distintos componentes del grafo de red social.

Nodos:

- Para referirnos a los usuarios de la red social vamos a emplear las variables u, v y w y en caso necesario, vamos a emplear subíndices.
- Para referirnos al conjunto de todos los usuarios de la red social de entrenamiento vamos a emplear el símbolo U.

Aristas: Una arista representa un vínculo entre dos usuarios en el grafo de red social.

- Para referirnos a las aristas vamos a seguir la notación (u, v) siendo u el usuario que actúa sobre el usuario v en el sentido de realizar una acción social que referencie al usuario v sin necesidad de que el usuario v intervenga en la misma de forma activa. Es importante tener presente que el grafo con el que trabajamos es un grafo dirigido, de modo que $(u, v) \neq (v, u)$. Es decir, el orden en el que nombremos los usuarios forma parte de la información que aporta la arista:
  - En el caso de una interacción de tipo *follow*, (u, v) expresa que el usuario u sigue al usuario v. El usuario v puede seguir o no seguir al usuario u.
  - En el caso de una interacción de tipo *reply*, (u, v) expresa que el usuario u ha contestado al menos a un tweet cuyo autor es el usuario v.
  - En el caso de una interacción de tipo *retweet*, (u, v) expresa que el usuario u ha *retwitteado* al menos un *tweet* cuyo autor es el usuario v.
  - En el caso de una interacción de tipo *mention*, (u, v) expresa que el usuario u ha mencionado en al menos un *tweet* al usuario v.

En lo sucesivo, para facilitar el la comprensión de los algoritmos presentados, vamos a aludir a las interacciones sociales dirigidas verbalmente como u sigue a v.

- Para referirnos al conjunto de todas las aristas en el grafo de red social de entrenamiento vamos a emplear el símbolo A.

Scores: Cada recomendador va a ser descrito formalmente con una función generadora de score que vamos a llamar s.

$$s : U \rightarrow [0, +\infty) \tag{4}$$

Esta función en los casos en los que dependa del usuario para el cual se generan las recomendaciones se va a denotar como s_v siendo v el usuario para el cual se generan recomendaciones personalizadas.

$$s_v : U \rightarrow [0, +\infty) \tag{5}$$

Para ilustrar el funcionamiento de los algoritmos presentados vamos a reproducir el funcionamiento de los mismos utilizando el grafo de entrenamiento definido como:

$$U = \{u_1, u_2, u_3, u_4, u_5, u_6, u_7, u_8, u_9, u_{10}\}$$

$$A = \{(u_2, u_1), (u_3, u_1), (u_3, u_2), (u_3, u_4), (u_3, u_5), (u_3, u_6),$$
$$(u_4, u_1), (u_5, u_2), (u_6, u_5), (u_6, u_9), (u_7, u_3), (u_7, u_4), (u_7, u_6),$$
$$(u_7, u_8), (u_8, u_6), (u_8, u_1), (u_9, u_5), (u_9, u_6), (u_1, u_6), (u_1, u_9)\}$$

La figura 23 muestra el aspecto del grafo descrito sobre estas líneas.

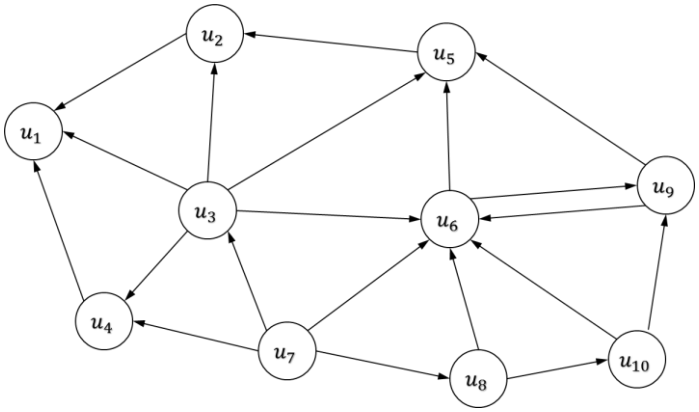


Figura 23. Representación gráfica del grafo de entrenamiento de ejemplo.

A continuación se presentan los algoritmos de recomendación basados en la topología de red social del grafo que representa las interacciones entre los usuarios en términos de la notación establecida.

5.1.1 Popularity

Vamos a definir la popularidad de un usuario dentro de un grafo dirigido como su *grado entrante* es decir, el número de aristas del grafo de red social entrantes en el nodo que representa al usuario; dicho en terminología Twitter, el número de seguidores.

El algoritmo de recomendación *Popularity Recommender* no es personalizado, es decir, va a asignar a cada usuario u susceptible de ser recomendado al usuario v un *score* independientemente de qué usuario sea v ($s_v(u) = s_w(u) = s(u)\ \forall\ u, v, w \in U$).

Por tanto este algoritmo queda definido por la función *score* s que asigna a cada usuario como *score* su *in-degree*:

$$s(u) = \sum_{\{v \in U : v \neq u\}} \delta_u(v) \tag{6}$$

Donde la función δ_u se define como:

$$\delta_u(v) = \begin{cases} 1 \text{ si } (v,u) \in A \\ 0 \text{ en caso contrario} \end{cases} \tag{7}$$

Podemos calcular s en todos los nodos del grafo de ejemplo para ver cuáles son los usuarios que aparecerán mejor posicionados y peor posicionados en las listas de recomendación. Para ello basta contar el número de aristas entrantes en cada nodo del grafo.

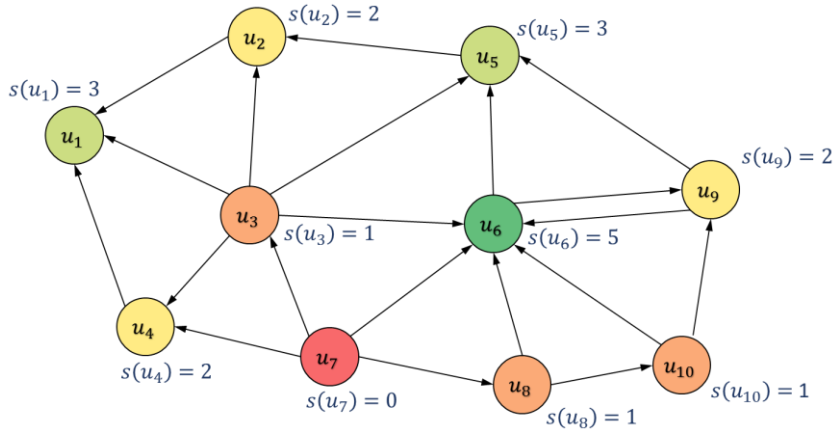


Figura 24. Actuación del algoritmo Popularity Recommender en el grafo de entrenamiento de ejemplo.

Como se puede observar en la figura 24, el usuario u_6 es el mejor posicionado en las recomendaciones mientras que el usuario u_7 resulta el peor posicionado.

La tabla 2 presenta las listas recomendaciones que genera el sistema de recomendación haciendo uso de *Popularity Recommender* para los distintos usuarios solicitantes en forma de tabla siendo la cabecera de las columnas el usuario solicitante de una lista de recomendación y el contenido de las columnas, la lista de recomendaciones generadas para el usuario cabecera de la columna:

u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9	u_{10}
u_6	u_6	u_9	u_6	u_6	u_1	u_1	u_1	u_1	u_1
u_5	u_5	u_8	u_5	u_1	u_2	u_5	u_5	u_2	u_5
u_2	u_4	u_{10}	u_2	u_4	u_4	u_2	u_2	u_4	u_2
u_4	u_9	u_7	u_9	u_9	u_3	u_9	u_4	u_3	u_4
u_9	u_3		u_3	u_3	u_8	u_{10}	u_9	u_8	u_3
u_3	u_8		u_8	u_8	u_{10}		u_3	u_{10}	u_8
u_8	u_{10}		u_{10}	u_{10}	u_7		u_7	u_7	u_7
u_{10}	u_7		u_7	u_7					
u_7									

Tabla 2. Recomendaciones generadas por el sistema de recomendación utilizando el algoritmo Popularity Recommender tomando como datos de entrenamiento el grafo de ejemplo.

Como podemos observar, dado que $s_v(u) = s_w(u) = s(u)\ \forall\ u, v, w \in U$, todas las listas de recomendación son el resultado de excluir de la lista de usuarios ordenados por *in-degree* $\{u_6, u_1, u_5, u_2, u_4, u_9, u_3, u_8, u_{10}, u_7\}$ el usuario objetivo (el usuario al cual está dirigida la lista de recomendación) y los usuarios sobre los que éste ya actúa. Esto último se debe a que el propio usuario y los usuarios que el usuario ya sigue no son susceptibles de ser recomendados y, por tanto, el sistema de recomendación no tomará a esos usuarios ni sus *score* en consideración a la hora de generar las listas de recomendaciones personalizadas.

5.1.2 MaxFOAF

El nombre de algoritmo *MaxFOAF Recommender* abrevia en inglés *Maximize Friends of a Friend* (Maximizar amigos de un amigo). Se trata de un algoritmo personalizado que depende esencialmente de la estructura de grafo dirigido de la red social. Esto se debe a que el criterio que emplea para asignar a cada usuario u susceptible de ser recomendado al usuario v un score se basa en el número de usuarios que sigue v, que a su vez siguen a u. Es decir, el concepto que subyace a este algoritmo es el de recomendar a los amigos de los amigos.

Formalmente el algoritmo de recomendación *MaxFOAF* queda definido por la función de *score* personalizada s_v definida como:

$$s_v(u) = \sum_{\{w \in U : \delta_w(v)=1\}} \delta_u(w) \tag{8}$$

Donde la función δ_u es la que se ha descrito previamente en la ecuación (7).

A modo de ejemplo podemos calcular s_{u_7} en todos los nodos del grafo de ejemplo para ver cuáles son los usuarios que aparecerán mejor y peor posicionados en las lista de recomendación que se va a generar para el usuario u_7:

La figura 25 muestra que los usuarios sobre los que sigue el usuario u_7 ($\{w \in U : \delta_w(u_7) = 1\}$) son $\{u_3, u_4, u_6, u_8\}$.

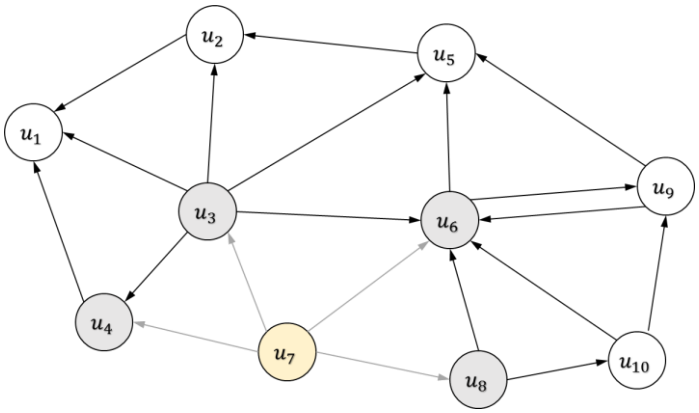


Figura 25. Usuarios sobre los que actúa el usuario u_7 en el grafo de ejemplo.

Aquellos usuarios que van a recibir scores positivos ejecutando el algoritmo *MaxFOAF Recommender* para generar la lista de recomendaciones para el usuario u_7 son los usuarios que sigue al menos uno de los usuarios que sigue el usuario u_7, sin incluir a aquellos que ya seguía u_7, que no son susceptibles de ser recomendados a u_7.

Es decir, son $\{u_1, u_2, u_4, u_5, u_6, u_9, u_{10}\} \setminus \{u_4, u_6\} = \{u_1, u_2, u_5, u_9, u_{10}\}$. La figura 26 muestra estos usuarios resaltados en verde claro.

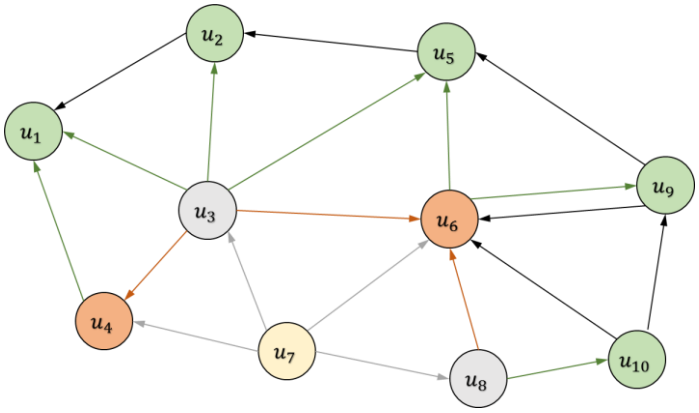


Figura 26. Usuarios sobre los que actúan los usuarios sobre los que actúa u_7 en el grafo de ejemplo.

La figura 27 muestra que el *score* que van a recibir estos usuarios es su *in-degree* en el subgrafo compuesto por las aristas salientes de los usuarios que no sigue el usuario u_7. El resto de usuarios, en caso de que los hubiera, no reciben *score*:

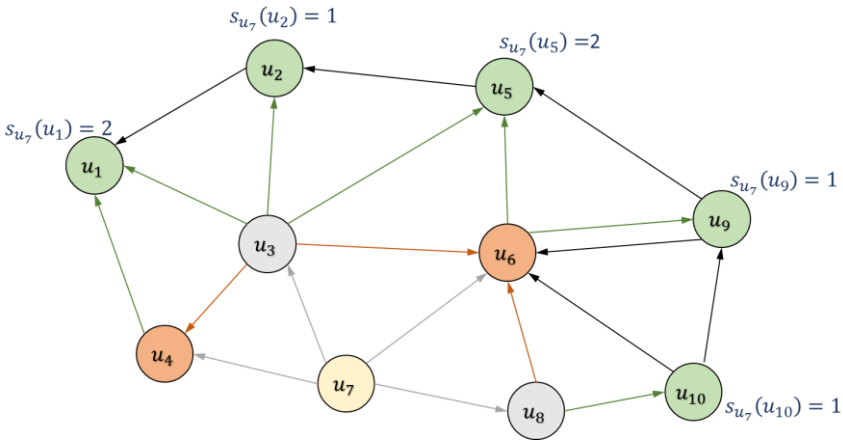


Figura 27. Actuación del algoritmo MaxFOAF Recommender en el grafo de entrenamiento de ejemplo para generar recomendaciones dirigidas al usuario u_7.

La tabla 3 presenta las listas de recomendaciones que va a generar el sistema de recomendación haciendo uso de *MaxFOAF Recommender* para los distintos usuarios solicitantes en forma de tabla siendo la cabecera de las columnas el usuario solicitante de una lista de recomendación y el contenido de las columnas, la lista de recomendaciones generadas para el usuario cabecera de la columna:

u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9	u_{10}
		u_9		u_1	u_2	u_1	u_5	u_2	u_5
						u_5	u_9		
						u_2			
						u_9			
						u_{10}			

Tabla 3. Recomendaciones generadas por el sistema de recomendación utilizando el algoritmo MaxFOAF Recommender tomando como datos de entrenamiento el grafo de ejemplo.

Como podemos observar si comparamos la cantidad de recomendaciones por usuario que ha generado *MaxFOAF Recommender* con las recomendaciones producidas por *Popularity Recommender* en la tabla 2, *MaxFOAF* es más discriminatorio que *Popularity* y

de hecho se hace patente el efecto negativo del arranque en frío o *cold start* para este algoritmo de recomendación: tiene problemas a la hora de generar recomendaciones a usuarios que no han sido suficientemente activos en interacción.

5.1.3 Random

El algoritmo *Random Recommender*, como su propio nombre indica, genera scores aleatorios para cada par usuario solicitante de recomendaciones, usuario susceptible de ser recomendado. Incluimos este método en nuestras pruebas a fin de tomarlo como referencia comparativa.

Formalmente el algoritmo de recomendación *Random Recommender* queda definido por la función de *score* s_v definida como:

$$s_v(u) = rand(0,1) \tag{9}$$

Donde $rand(0,1)$ representa un número pseudo-aleatorio en el intervalo semiabierto [0,1) con distribución de probabilidad uniforme. Este número lo obtenemos haciendo uso de la clase `java.util.Random` empleando como semilla el instante en el que se inicia el programa de generación de recomendaciones.

A modo de ejemplo podemos calcular s_{u_7} en todos los nodos del grafo de ejemplo sobre los que no actúa el usuario u_7 ($\{w \in U: \delta_w(u_7) = 0\} = \{u_1, u_2, u_5, u_7, u_9, u_{10}\}$):

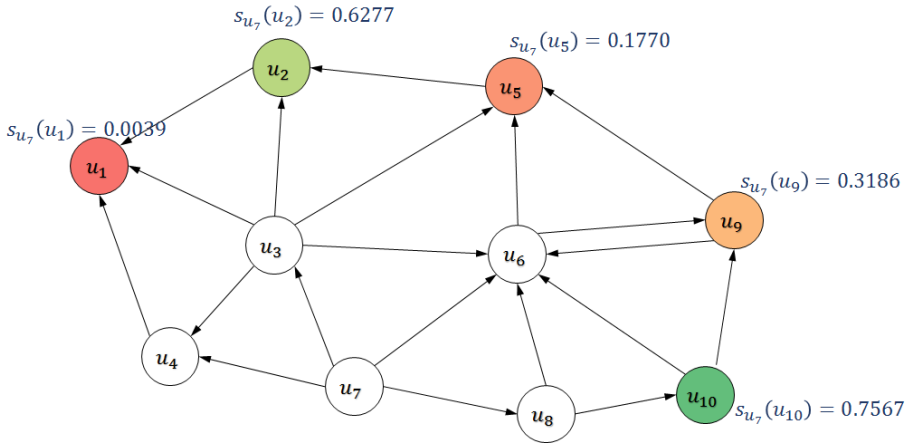


Figura 28. Actuación del algoritmo Random Recommender en el grafo de entrenamiento de ejemplo para generar recomendaciones dirigidas al usuario u_7.

La tabla 4 una de las posibles listas de recomendaciones que va a generar el sistema de recomendación haciendo uso de *Random Recommender* para los distintos usuarios solicitantes en forma de tabla siendo la cabecera de las columnas el usuario solicitante de una lista de recomendación y el contenido de las columnas, la lista de recomendaciones generadas para el usuario cabecera de la columna:

u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9	u_{10}
u_8	u_3	u_9	u_8	u_3	u_4	u_{10}	u_5	u_{10}	u_2
u_6	u_9	u_7	u_6	u_6	u_7	u_2	u_2	u_4	u_8
u_5	u_8	u_{10}	u_3	u_4	u_8	u_9	u_1	u_3	u_4

u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9	u_{10}
u_4	u_{10}	u_8	u_5	u_1	u_3	u_5	u_3	u_2	u_3
u_9	u_6		u_7	u_{10}	u_1	u_1	u_7	u_7	u_5
u_7	u_4		u_9	u_7	u_{10}		u_9	u_1	u_7
u_2	u_5		u_{10}	u_9	u_2		u_4	u_8	u_1
u_3	u_7		u_2	u_8					
u_{10}									

Tabla 4. Recomendaciones generadas por el sistema de recomendación utilizando el algoritmo Random Recommender tomando como datos de entrenamiento el grafo de ejemplo.

Este algoritmo constituye una línea base trivial que se emplea como referencia de la mínima precisión posible que se puede llegar a obtener a la hora de generar recomendaciones. No obstante, a la hora de considerar métricas alternativas tales como novedad o diversidad de las recomendaciones, este algoritmo obtiene los mejores resultados. A la hora de buscar algoritmos que mejoren la novedad y diversidad de las recomendaciones, el objetivo va a ser alejarse de los resultados de este recomendador en lo que respecta a precisión y acercarse a los resultados de este recomendador en lo que respecta a novedad y diversidad.

5.1.4 PageRank

El algoritmo *PageRank*, publicado por Lawrence Page y Sergey Brin (1998) modela las probabilidades de un caminante aleatorio en el grafo, de encontrarse en cada nodo del mismo en un instante dado. Calculando *PageRank* en cada uno de los nodos del grafo se valora por una parte la cantidad de aristas entrantes en cada nodo, por otra parte la relevancia de los nodos de los cuales provienen las aristas entrantes en cada nodo y finalmente se valora inversamente la cantidad de aristas salientes de los nodos de los cuales provienen las aristas entrantes en cada nodo.

Este recomendador, al igual que sucedía con el recomendador *Popularity Recommender* no es personalizado, sino que asigna a cada usuario u susceptible de ser recomendado al usuario v un *score* (el valor *PageRank* del nodo v) independientemente de qué usuario sea v ($s_v(u) = s_w(u) = s(u)\ \forall\, u, v, w \in U$).

Por tanto este algoritmo queda definido por la función *score* s que asigna a cada usuario como *score* su valor de *PageRank*:

$$s(u) = \frac{\alpha}{|U|} + (1-\alpha) \sum_{\{v \in U : \delta_u(v)\}} \frac{s(v)}{|\{w \in U : \delta_w(v)\}|} \tag{10}$$

Donde la función δ_u es la que se ha descrito previamente en la ecuación (7) y $\alpha \in (0,1)$ es el factor de teleportación que modela la probabilidad del caminante aleatorio de trasladarse a un nodo aleatorio del grafo aunque no exista una arista que una el nodo en el que el caminante se encuentra con el nodo del grafo al que se teleporta.

Como podemos observar, dado que s aparece tanto a la izquierda como a la derecha de la igualdad, lo que se nos plantea dada esta formulación de la función de score, es un sistema de ecuaciones con tantas ecuaciones e incógnitas como nodos encontramos en el

grafo de red social. Por suerte, asegurando que el factor de teleportación α no sea cero y tratando los nodos sumidero (aquellos que no tienen ninguna arista saliente) añadiéndoles una arista saliente hacia todos los nodos del grafo conseguimos que la matriz que modela este sistema de ecuaciones sea una matriz no negativa e irreducible en la que el método numérico de las potencias converge de forma razonable (Page & Brin 1998, Fernández 2004).

Podemos calcular *s* en todos los nodos del grafo de ejemplo para ver cuáles son los usuarios que aparecerán mejor posicionados y peor posicionados en las listas de recomendación. Para ello asignamos un *score PageRank* inicial a cada nodo $s_0 = \frac{1}{|U|} = \frac{1}{10}$ e iteramos actualizando los *score PageRank* de cada nodo empleando para ello la ecuación (10) tomando a la derecha de la igualdad los score *PageRank* de la iteración anterior tal y como se muestra en la tabla 5.

	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9	u_{10}
0	0.1000	0.1000	0.1000	0.1000	0.1000	0.1000	0.1000	0.1000	0.1000	0.1000
1	0.2040	0.1240	0.0480	0.0640	0.1080	0.1840	0.0280	0.0480	0.0680	0.0680
2	0.1944	0.1304	0.0419	0.0496	0.1371	0.1232	0.0363	0.0419	0.0635	0.0555
3	0.1863	0.1520	0.0428	0.0495	0.1102	0.1139	0.0356	0.0428	0.0578	0.0523
4	0.2029	0.1299	0.0420	0.0489	0.1036	0.1100	0.0349	0.0420	0.0558	0.0520
5	0.1860	0.1258	0.0432	0.0499	0.1026	0.1099	0.0362	0.0432	0.0570	0.0530
6	0.1824	0.1239	0.0421	0.0490	0.1017	0.1104	0.0349	0.0421	0.0561	0.0522
7	0.1796	0.1227	0.0416	0.0483	0.1012	0.1085	0.0346	0.0416	0.0555	0.0514
8	0.1778	0.1220	0.0413	0.0479	0.0999	0.1073	0.0344	0.0413	0.0549	0.0510
9	0.1768	0.1208	0.0411	0.0477	0.0991	0.1066	0.0342	0.0411	0.0546	0.0507
10	0.1755	0.1200	0.0410	0.0476	0.0986	0.1061	0.0341	0.0410	0.0544	0.0506
11	0.1747	0.1195	0.0409	0.0474	0.0983	0.1058	0.0340	0.0409	0.0543	0.0504
12	0.1741	0.1191	0.0408	0.0473	0.0980	0.1056	0.0340	0.0408	0.0541	0.0503
13	0.1736	0.1189	0.0407	0.0472	0.0978	0.1053	0.0339	0.0407	0.0541	0.0502
14	0.1733	0.1186	0.0407	0.0472	0.0976	0.1052	0.0339	0.0407	0.0540	0.0502
15	0.1730	0.1185	0.0406	0.0471	0.0975	0.1051	0.0339	0.0406	0.0539	0.0501
16	0.1729	0.1184	0.0406	0.0471	0.0974	0.1050	0.0338	0.0406	0.0539	0.0501
17	0.1727	0.1183	0.0406	0.0471	0.0974	0.1049	0.0338	0.0406	0.0539	0.0501
18	0.1726	0.1182	0.0406	0.0471	0.0973	0.1049	0.0338	0.0406	0.0538	0.0501
19	0.1725	0.1182	0.0406	0.0471	0.0973	0.1049	0.0338	0.0406	0.0538	0.0500
20	0.1725	0.1181	0.0406	0.0471	0.0973	0.1048	0.0338	0.0406	0.0538	0.0500
21	0.1724	0.1181	0.0406	0.0471	0.0973	0.1048	0.0338	0.0406	0.0538	0.0500
22	0.1724	0.1181	0.0406	0.0470	0.0972	0.1048	0.0338	0.0406	0.0538	0.0500
23	0.1724	0.1181	0.0406	0.0470	0.0972	0.1048	0.0338	0.0406	0.0538	0.0500
24	0.1724	0.1181	0.0406	0.0470	0.0972	0.1048	0.0338	0.0406	0.0538	0.0500
25	0.1724	0.1181	0.0405	0.0470	0.0972	0.1048	0.0338	0.0405	0.0538	0.0500
26	0.1724	0.1181	0.0405	0.0470	0.0972	0.1048	0.0338	0.0405	0.0538	0.0500
27	0.1723	0.1181	0.0405	0.0470	0.0972	0.1048	0.0338	0.0405	0.0538	0.0500
28	0.1723	0.1180	0.0405	0.0470	0.0972	0.1048	0.0338	0.0405	0.0538	0.0500
29	0.1723	0.1180	0.0405	0.0470	0.0972	0.1048	0.0338	0.0405	0.0538	0.0500

Tabla 5. Evolución iterativa de los score PageRank de cada nodo del grafo de ejemplo iterando según el método numérico de las potencias con factor de teleportación $\alpha = 0.2$ con tratamiento de nodos sumidero que hace que el usuario u_1 actúe como si actuase sobre todos los usuarios incluido él mismo.

De modo que el resultado del cálculo de *PageRank* en el grafo de ejemplo es el que se muestra en la figura 29.

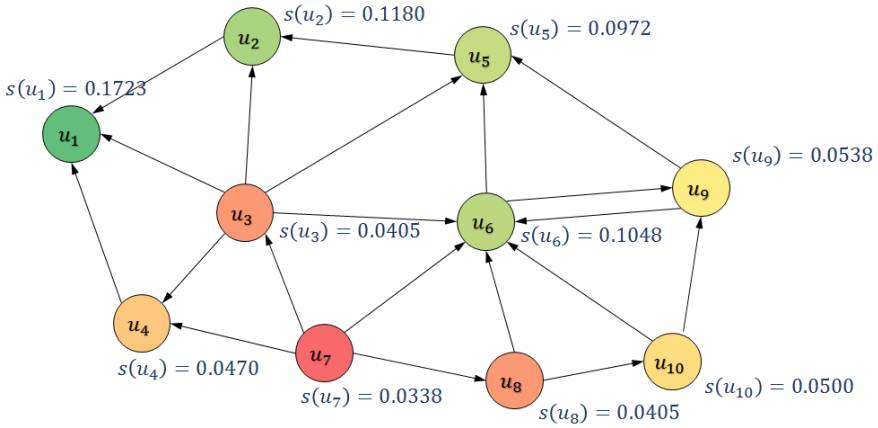


Figura 29. Actuación del algoritmo PageRank en el grafo de entrenamiento de ejemplo.

La tabla 6 presenta las listas recomendaciones que genera el sistema de recomendación haciendo uso de *PageRank* para los distintos usuarios solicitantes de recomendaciones:

u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9	u_{10}
u_2	u_6	u_9	u_2	u_1	u_1	u_1	u_1	u_1	u_1
u_6	u_5	u_{10}	u_6	u_6	u_2	u_2	u_2	u_2	u_2
u_5	u_9	u_8	u_5	u_9	u_{10}	u_5	u_5	u_{10}	u_5
u_9	u_{10}	u_7	u_9	u_{10}	u_4	u_9	u_9	u_4	u_4
u_{10}	u_4		u_{10}	u_4	u_3	u_{10}	u_4	u_3	u_3
u_4	u_3		u_3	u_3	u_8		u_3	u_8	u_8
u_3	u_8		u_8	u_8	u_7		u_7	u_7	u_7
u_8	u_7		u_7	u_7					
u_7									

Tabla 6. Recomendaciones generadas por el sistema de recomendación utilizando el algoritmo PageRank, con factor de teleportación $\alpha = 0.2$, tomando como datos de entrenamiento el grafo de ejemplo.

Como podemos observar el usuario u_1 es el mejor posicionado en las recomendaciones a pesar de no ser el nodo con mayor *in-degree*. El usuario u_7 es el peor posicionado.

Al igual que sucedía con el recomendador no personalizado *Popularity Recommender*, dado que $s_v(u) = s_w(u) = s(u)\ \forall\ u, v, w \in U$, todas las listas de recomendación son el resultado de excluir de la lista de usuarios ordenados por *PageRank* $\{u_1, u_2, u_6, u_5, u_9, u_{10}, u_4, u_3, u_8, u_7\}$ el usuario al cual está dirigida la lista de recomendación y los usuarios sobre los que éste ya actúa.

5.1.5 PageRank personalizado

El algoritmo *PageRank* presentado anteriormente genera un ranking común para todos los usuarios y por sí solo no es capaz por tanto de generar recomendaciones personalizadas para los usuarios.De cara a aprovechar el potencial de *PageRank* para generar recomendaciones personalizadas utilizams una versión donde se modifica la distribución de probabilidad de los distintos nodos en el suceso aleatorio que se ha definido como teleportación (el caminante aleatorio se traslada esporádicamente con probabilidad α a un nodo aleatorio), tal y como se propone en (White & Smyth, 2003).

En particular, la distribución escogida, personalizada para cada usuario susceptible de recibir recomendaciones, asigna al nodo que representa al usuario para el cual se están generando recomendaciones probabilidad 1 de ser el destino de cualquier teleportación del caminante aleatorio y asigna probabilidad 0 a cualquier otro nodo del grafo.

Por tanto este algoritmo queda definido por la función *score* personalizada s_v definida como:

$$s_v(u) = \begin{cases} (1-\alpha) \sum\limits_{\{w \in U: \delta_u(w)\}} \dfrac{s(w)}{|\{w' \in U: \delta_{w'}(w)\}|} & \text{si } u \neq v \\ \alpha + (1-\alpha) \sum\limits_{\{w \in U: \delta_u(w)\}} \dfrac{s(w)}{|\{w' \in U: \delta_{w'}(w)\}|} & \text{si } u = v \end{cases} \tag{11}$$

De este modo se da prioridad al vecindario más próximo al usuario para el cual se están generando las recomendaciones a la hora de generar el ranking *PageRank* de los nodos del grafo, personalizando así las recomendaciones para cada usuario presente en la red social.

A modo de ejemplo podemos calcular s_{u_7} en todos los nodos del grafo de ejemplo para ver cuáles son los usuarios que aparecerán mejor posicionados y peor posicionados en las lista de recomendación que se va a generar para el usuario u_7. Para ello asignamos un *score PageRank* inicial a cada nodo $s_0 = \frac{1}{|U|} = \frac{1}{10}$ e iteramos actualizando los *score* personalizados de cada nodo empleando para ello la ecuación (11) tomando a la derecha de la igualdad los *score* personalizados de la iteración anterior tal y como se muestra en la tabla 7.

	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9	u_{10}
0	0.1000	0.1000	0.1000	0.1000	0.1000	0.1000	0.1000	0.1000	0.1000	0.1000
1	0.1840	0.1040	0.0280	0.0440	0.0880	0.1640	0.2080	0.0280	0.0480	0.0480
2	0.1376	0.0896	0.0563	0.0608	0.0995	0.1104	0.2147	0.0563	0.0339	0.0259
3	0.1403	0.0996	0.0540	0.0630	0.0687	0.1094	0.2110	0.0540	0.0214	0.0335
4	0.1499	0.0748	0.0534	0.0621	0.0635	0.1056	0.2112	0.0534	0.0246	0.0328
5	0.1301	0.0714	0.0542	0.0628	0.0641	0.1071	0.2120	0.0542	0.0251	0.0334
6	0.1264	0.0704	0.0528	0.0615	0.0633	0.1066	0.2104	0.0528	0.0238	0.0321
7	0.1240	0.0692	0.0522	0.0606	0.0622	0.1041	0.2101	0.0522	0.0230	0.0312
8	0.1222	0.0681	0.0519	0.0603	0.0607	0.1029	0.2099	0.0519	0.0224	0.0308
9	0.1208	0.0667	0.0518	0.0601	0.0599	0.1021	0.2098	0.0518	0.0221	0.0306
10	0.1193	0.0658	0.0516	0.0599	0.0594	0.1017	0.2097	0.0516	0.0219	0.0304
11	0.1184	0.0653	0.0515	0.0597	0.0590	0.1013	0.2095	0.0515	0.0217	0.0302
12	0.1177	0.0649	0.0514	0.0596	0.0587	0.1010	0.2095	0.0514	0.0215	0.0301
13	0.1172	0.0646	0.0513	0.0595	0.0584	0.1007	0.2094	0.0513	0.0214	0.0300
14	0.1169	0.0643	0.0513	0.0595	0.0582	0.1006	0.2094	0.0513	0.0214	0.0299
15	0.1166	0.0642	0.0512	0.0594	0.0581	0.1004	0.2093	0.0512	0.0213	0.0299
16	0.1164	0.0640	0.0512	0.0594	0.0580	0.1004	0.2093	0.0512	0.0213	0.0298
17	0.1162	0.0639	0.0512	0.0594	0.0580	0.1003	0.2093	0.0512	0.0212	0.0298
18	0.1161	0.0639	0.0512	0.0593	0.0579	0.1002	0.2093	0.0512	0.0212	0.0298
19	0.1160	0.0638	0.0511	0.0593	0.0579	0.1002	0.2093	0.0511	0.0212	0.0298
20	0.1160	0.0638	0.0511	0.0593	0.0578	0.1002	0.2093	0.0511	0.0212	0.0297
21	0.1159	0.0637	0.0511	0.0593	0.0578	0.1001	0.2093	0.0511	0.0212	0.0297

	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9	u_{10}
22	0.1159	0.0637	0.0511	0.0593	0.0578	0.1001	0.2093	0.0511	0.0212	0.0297
23	0.1159	0.0637	0.0511	0.0593	0.0578	0.1001	0.2093	0.0511	0.0212	0.0297
24	0.1159	0.0637	0.0511	0.0593	0.0578	0.1001	0.2093	0.0511	0.0212	0.0297
25	0.1158	0.0637	0.0511	0.0593	0.0578	0.1001	0.2093	0.0511	0.0212	0.0297
26	0.1158	0.0637	0.0511	0.0593	0.0578	0.1001	0.2093	0.0511	0.0212	0.0297
27	0.1158	0.0637	0.0511	0.0593	0.0578	0.1001	0.2093	0.0511	0.0212	0.0297
28	0.1158	0.0637	0.0511	0.0593	0.0578	0.1001	0.2093	0.0511	0.0212	0.0297
29	0.1158	0.0637	0.0511	0.0593	0.0578	0.1001	0.2093	0.0511	0.0212	0.0297

Tabla 7. Evolución iterativa de los score PageRank Personalizados para el usuario u_7 de cada nodo del grafo de ejemplo iterando según el método numérico de las potencias con factor de teleportación $\alpha = 0.2$ con tratamiento de nodos sumidero que hace que el usuario u_1 actúe como si actuase sobre todos los usuarios incluido él mismo.

De modo que el resultado del cálculo de *PageRank* personalizado orientado al usuario u_7 en el grafo de ejemplo es el que se muestra en la figura 30.

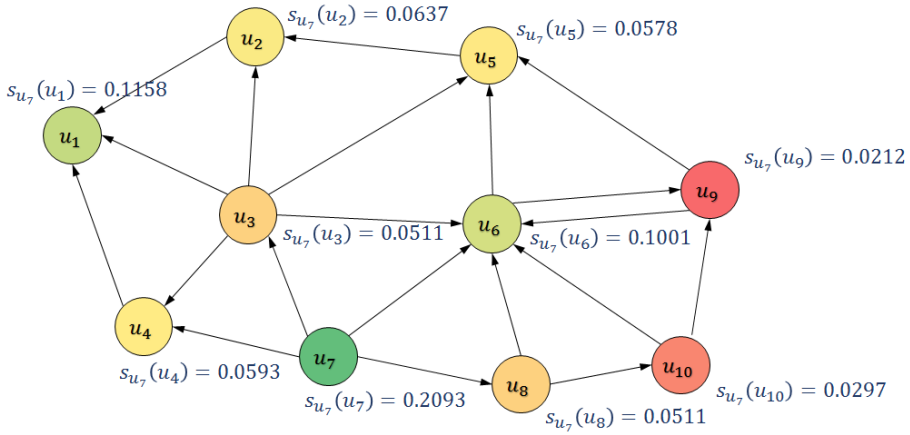


Figura 30. Actuación del algoritmo Personalized PageRank Recommender en el grafo de entrenamiento de ejemplo orientado al usuario u_7.

A continuación presentamos las listas recomendaciones que genera el sistema de recomendación haciendo uso de *Personalized PageRank Recommender* para los distintos usuarios solicitantes de recomendaciones:

u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9	u_{10}
u_2	u_6	u_9	u_2	u_1	u_2	u_1	u_5	u_2	u_5
u_6	u_5	u_{10}	u_6	u_6	u_1	u_2	u_2	u_1	u_2
u_5	u_9	u_8	u_5	u_9	u_{10}	u_5	u_1	u_{10}	u_1
u_9	u_{10}	u_7	u_9	u_{10}	u_4	u_{10}	u_9	u_4	u_4
u_{10}	u_4		u_{10}	u_4	u_3	u_9	u_4	u_3	u_3
u_4	u_3		u_3	u_3	u_8		u_3	u_8	u_8
u_3	u_8		u_8	u_8	u_7		u_7	u_7	u_7
u_8	u_7		u_7	u_7					
u_7									

Tabla 8. Recomendaciones generadas por el sistema de recomendación utilizando el algoritmo Personalized PageRank Recommender, con factor de teleportación $\alpha = 0.2$, tomando como datos de entrenamiento el grafo de ejemplo.

Este algoritmo constituye en definitiva una alternativa personalizada al algoritmo *PageRank* no personalizado sin alejarse de los criterios que definen *PageRank* a la hora de generar un ranking de usuarios: se valora por una parte la cantidad de aristas entrantes en cada nodo, por otra parte la relevancia de los nodos de los cuales provienen las aristas entrantes en cada nodo y finalmente se valora inversamente la cantidad de aristas salientes de los nodos de los cuales provienen las aristas entrantes en cada nodo.

5.1.6 HITS

El algoritmo *HITS (Hyperlink-Induced Topic Search)*, publicado por Jon Kleinberg en (Kleinberg, 1999) juega con los conceptos de *autoridad* y *hub* (centro de actividad) que se relacionan complementariamente: El grado de *autoridad* es proporcional a la suma del grado de *hub* de los nodos entrantes mientras que el grado de *hub* es proporcional a la suma de la *autoridad* de los nodos salientes.

Formalmente el algoritmo *HITS* queda definido por las funciones $authority$ y hub que asignan a cada usuario sus *score* de *autoridad* y *hub* cuyos dominios e imágenes coinciden con los dominios e imágenes de las funciones de *score* no personalizadas definidas en (4):

$$authority(u) = \sum_{\{v \in U : \delta_u(v)\}} hub(v) \tag{12}$$

$$hub(u) = \sum_{\{v \in U : \delta_v(u)\}} authority(v) \tag{13}$$

Donde la función δ_u es la que se ha descrito previamente en la ecuación (7).

Podemos sustituir en cada una de las dos ecuaciones la ecuación contraria llegando a las siguientes dos ecuaciones:

$$authority(u) = \sum_{\{v \in U : \delta_u(v)\}} \sum_{\{w \in U : \delta_w(v)\}} authority(w) \tag{14}$$

$$hub(u) = \sum_{\{v \in U : \delta_v(u)\}} \sum_{\{w \in U : \delta_v(w)\}} hub(w) \tag{15}$$

Como podemos observar, dado que $authority$ y hub aparecen en ambas igualdades a la izquierda y a la derecha, lo que se nos plantea dada esta formulación de la función de score, son dos sistemas de ecuaciones cada uno con tantas ecuaciones e incógnitas como el número de nodos que encontramos en el grafo de red social.

Dado que abordar estos sistemas de ecuaciones de forma exacta es computacionalmente muy costoso, Jon Kleinberg en (Kleinberg, 1999) propone un método numérico iterativo para encontrar una solución aproximada de forma iterativa para los dos sistemas

a la vez consistente en asignar un *score* inicial a cada nodo ($authority_0(u) = hub_0(u) = \frac{1}{\sqrt[2]{|U|}} \quad \forall u \in U$), actualizar los valores de *autoridad* y *hub* en cada iteración haciendo uso de las fórmulas (12) y (13) tomando a la derecha de las igualdades los valores de la iteración anterior y normalizando los nuevos valores asignados de forma que la norma euclídea de los vectores de *autoridad* y *hub* sea 1 en cada iteración, garantizando así la convergencia del método.

La tabla 9 muestra a modo de ejemplo vamos la ejecución este algoritmo en el grafo de ejemplo para ver cuáles son los *score* de *autoridad* y *hub* de cada nodo del mismo.

	Authority										Hub									
	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9	u_{10}	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9	u_{10}
0	0.3162	0.3162	0.3162	0.3162	0.3162	0.3162	0.3162	0.3162	0.3162	0.3162	0.3162	0.3162	0.3162	0.3162	0.3162	0.3162	0.3162	0.3162	0.3162	0.3162
1	1.2649	0.6325	0.3162	0.6325	0.9487	1.5811	0.0000	0.3162	0.6325	0.3162	0.0000	0.3162	1.5811	0.3162	0.3162	0.6325	1.2649	0.6325	0.6325	0.6325
	0.4961	0.2481	0.1240	0.2481	0.3721	0.6202	0.0000	0.1240	0.2481	0.1240	0.0000	0.1291	0.6455	0.1291	0.1291	0.2582	0.5164	0.2582	0.2582	0.2582
2	0.9037	0.7746	0.5164	1.1619	1.1619	1.9365	0.0000	0.5164	0.5164	0.2582	0.0000	0.4961	1.9846	0.4961	0.2481	0.3772	1.1163	0.8784	0.9923	0.8682
	0.3058	0.2621	0.1747	0.3932	0.3932	0.6553	0.0000	0.1747	0.1747	0.0874	0.0000	0.1713	0.6851	0.1713	0.0856	0.1302	0.3854	0.3032	0.3426	0.2997
3	1.0277	0.7708	0.3854	1.0705	1.1579	2.0160	0.0000	0.3854	0.4299	0.3032	0.0000	0.3058	2.0095	0.3058	0.2621	0.2604	1.3979	0.9550	1.0484	0.8300
	0.3469	0.2602	0.1301	0.3614	0.3909	0.6806	0.0000	0.1301	0.1451	0.1024	0.0000	0.1018	0.6692	0.1018	0.0873	0.0867	0.4656	0.3181	0.3492	0.2764
4	0.8729	0.7565	0.4656	1.1348	1.1051	2.0784	0.0000	0.4656	0.3631	0.3181	0.0000	0.3469	2.0399	0.3469	0.2602	0.2324	1.3021	0.9570	1.0714	0.8257
	0.2938	0.2546	0.1567	0.3819	0.3719	0.6995	0.0000	0.1567	0.1222	0.1070	0.0000	0.1159	0.6813	0.1159	0.0869	0.0776	0.4349	0.3196	0.3578	0.2758
5	0.9130	0.7682	0.4349	1.1162	1.1168	2.0694	0.0000	0.4349	0.3534	0.3196	0.0000	0.2938	2.0016	0.2938	0.2546	0.2091	1.3947	0.9752	1.0714	0.8217
	0.3077	0.2589	0.1466	0.3762	0.3764	0.6974	0.0000	0.1466	0.1191	0.1077	0.0000	0.0979	0.6670	0.0979	0.0848	0.0697	0.4647	0.3250	0.3570	0.2738
6	0.8627	0.7518	0.4647	1.1317	1.0936	2.0874	0.0000	0.4647	0.3435	0.3250	0.0000	0.3077	2.0165	0.3077	0.2589	0.2039	1.3667	0.9712	1.0738	0.8165
	0.2908	0.2534	0.1567	0.3815	0.3687	0.7037	0.0000	0.1567	0.1158	0.1095	0.0000	0.1026	0.6724	0.1026	0.0863	0.0680	0.4557	0.3238	0.3580	0.2722
7	0.8775	0.7587	0.4557	1.1280	1.0984	2.0821	0.0000	0.4557	0.3402	0.3238	0.0000	0.2908	1.9982	0.2908	0.2534	0.2021	1.3986	0.9760	1.0724	0.8195
	0.2959	0.2558	0.1536	0.3804	0.3704	0.7021	0.0000	0.1536	0.1147	0.1092	0.0000	0.0970	0.6662	0.0970	0.0845	0.0674	0.4663	0.3254	0.3575	0.2732
8	0.8601	0.7507	0.4663	1.1324	1.0911	2.0885	0.0000	0.4663	0.3406	0.3254	0.0000	0.2959	2.0045	0.2959	0.2558	0.1992	1.3897	0.9753	1.0724	0.8168
	0.2900	0.2531	0.1572	0.3819	0.3679	0.7043	0.0000	0.1572	0.1148	0.1097	0.0000	0.0986	0.6682	0.0986	0.0853	0.0664	0.4633	0.3251	0.3575	0.2723
9	0.8655	0.7535	0.4633	1.1315	1.0922	2.0865	0.0000	0.4633	0.3387	0.3251	0.0000	0.2900	1.9972	0.2900	0.2531	0.2001	1.4006	0.9765	1.0722	0.8191
	0.2919	0.2541	0.1562	0.3816	0.3683	0.7036	0.0000	0.1562	0.1142	0.1096	0.0000	0.0967	0.6658	0.0967	0.0844	0.0667	0.4669	0.3256	0.3575	0.2731
10	0.8592	0.7502	0.4669	1.1328	1.0900	2.0889	0.0000	0.4669	0.3398	0.3256	0.0000	0.2919	1.9995	0.2919	0.2541	0.1986	1.3977	0.9767	1.0719	0.8178
	0.2898	0.2530	0.1575	0.3820	0.3676	0.7044	0.0000	0.1575	0.1146	0.1098	0.0000	0.0973	0.6666	0.0973	0.0847	0.0662	0.4660	0.3256	0.3574	0.2726
11	0.8612	0.7513	0.4660	1.1325	1.0902	2.0881	0.0000	0.4660	0.3389	0.3256	0.0000	0.2898	1.9968	0.2898	0.2530	0.1993	1.4014	0.9771	1.0720	0.8190
	0.2904	0.2534	0.1571	0.3819	0.3676	0.7042	0.0000	0.1571	0.1143	0.1098	0.0000	0.0966	0.6657	0.0966	0.0843	0.0664	0.4672	0.3257	0.3574	0.2730
12	0.8589	0.7500	0.4672	1.1329	1.0895	2.0891	0.0000	0.4672	0.3395	0.3257	0.0000	0.2904	1.9976	0.2904	0.2534	0.1986	1.4004	0.9773	1.0719	0.8185
	0.2896	0.2529	0.1576	0.3821	0.3674	0.7045	0.0000	0.1576	0.1145	0.1099	0.0000	0.0968	0.6659	0.0968	0.0845	0.0662	0.4669	0.3258	0.3573	0.2729
13	0.8596	0.7504	0.4669	1.1328	1.0895	2.0888	0.0000	0.4669	0.3391	0.3258	0.0000	0.2896	1.9966	0.2896	0.2529	0.1990	1.4017	0.9774	1.0719	0.8190
	0.2899	0.2531	0.1574	0.3820	0.3674	0.7044	0.0000	0.1574	0.1144	0.1099	0.0000	0.0966	0.6656	0.0966	0.0843	0.0663	0.4673	0.3258	0.3574	0.2730
14	0.8587	0.7499	0.4673	1.1329	1.0893	2.0892	0.0000	0.4673	0.3394	0.3258	0.0000	0.2899	1.9969	0.2899	0.2531	0.1987	1.4014	0.9775	1.0719	0.8188
	0.2896	0.2529	0.1576	0.3821	0.3674	0.7046	0.0000	0.1576	0.1144	0.1099	0.0000	0.0966	0.6657	0.0966	0.0844	0.0662	0.4672	0.3259	0.3573	0.2730
15	0.8590	0.7501	0.4672	1.1329	1.0893	2.0891	0.0000	0.4672	0.3392	0.3259	0.0000	0.2896	1.9965	0.2896	0.2529	0.1988	1.4018	0.9775	1.0719	0.8190
	0.2897	0.2530	0.1576	0.3821	0.3674	0.7045	0.0000	0.1576	0.1144	0.1099	0.0000	0.0965	0.6656	0.0965	0.0843	0.0663	0.4673	0.3259	0.3574	0.2730
16	0.8587	0.7499	0.4673	1.1329	1.0892	2.0892	0.0000	0.4673	0.3393	0.3259	0.0000	0.2897	1.9966	0.2897	0.2530	0.1987	1.4017	0.9776	1.0719	0.8189
	0.2896	0.2529	0.1576	0.3821	0.3673	0.7046	0.0000	0.1576	0.1144	0.1099	0.0000	0.0966	0.6656	0.0966	0.0843	0.0662	0.4673	0.3259	0.3573	0.2730
17	0.8588	0.7499	0.4673	1.1329	1.0892	2.0892	0.0000	0.4673	0.3393	0.3259	0.0000	0.2896	1.9965	0.2896	0.2529	0.1988	1.4018	0.9776	1.0719	0.8190
	0.2896	0.2529	0.1576	0.3821	0.3673	0.7046	0.0000	0.1576	0.1144	0.1099	0.0000	0.0965	0.6656	0.0965	0.0843	0.0663	0.4673	0.3259	0.3573	0.2730
18	0.8587	0.7499	0.4673	1.1329	1.0892	2.0892	0.0000	0.4673	0.3393	0.3259	0.0000	0.2896	1.9965	0.2896	0.2529	0.1987	1.4018	0.9776	1.0719	0.8190
	0.2896	0.2529	0.1576	0.3821	0.3673	0.7046	0.0000	0.1576	0.1144	0.1099	0.0000	0.0966	0.6656	0.0966	0.0843	0.0662	0.4673	0.3259	0.3573	0.2730
19	0.8587	0.7499	0.4673	1.1329	1.0892	2.0892	0.0000	0.4673	0.3393	0.3259	0.0000	0.2896	1.9964	0.2896	0.2529	0.1987	1.4019	0.9776	1.0719	0.8190
	0.2896	0.2529	0.1576	0.3821	0.3673	0.7046	0.0000	0.1576	0.1144	0.1099	0.0000	0.0965	0.6656	0.0965	0.0843	0.0663	0.4674	0.3259	0.3573	0.2730
20	0.8586	0.7499	0.4674	1.1329	1.0892	2.0892	0.0000	0.4674	0.3393	0.3259	0.0000	0.2896	1.9965	0.2896	0.2529	0.1987	1.4019	0.9776	1.0719	0.8190
	0.2896	0.2529	0.1576	0.3821	0.3673	0.7046	0.0000	0.1576	0.1144	0.1099	0.0000	0.0965	0.6656	0.0965	0.0843	0.0663	0.4673	0.3259	0.3573	0.2730

Tabla 9. Evolución iterativa de los score HITS Authority y Hub en cada nodo del grafo de ejemplo iterando según el método numérico propuesto en (Kleinberg, 1999).

Este algoritmo de análisis de grafos da lugar a dos recomendadores no personalizados: *HITS Authority* y *HITS Hub*.

5.1.6.1 HITS Authority

El algoritmo de recomendación no personalizado *HITS Authority* asigna a cada usuario como *score*, independientemente del usuario que solicite las recomendaciones, el *score*

authority que se asigna al nodo que representa al usuario susceptible de ser recomendado iterando el algoritmo HITS en el grafo que representa la red social.

Por tanto este algoritmo queda definido por la función *score* s que asigna a cada usuario como *score* su *score authority*:

$$s(u) = authority(u) \tag{16}$$

Donde $authority$ es la función presentada en (12).

Si nos fijamos en la primera mitad de la última fila de la tabla 9 podemos ver cuál es la actuación de este recomendador sobre el grafo de ejemplo (figura 31).

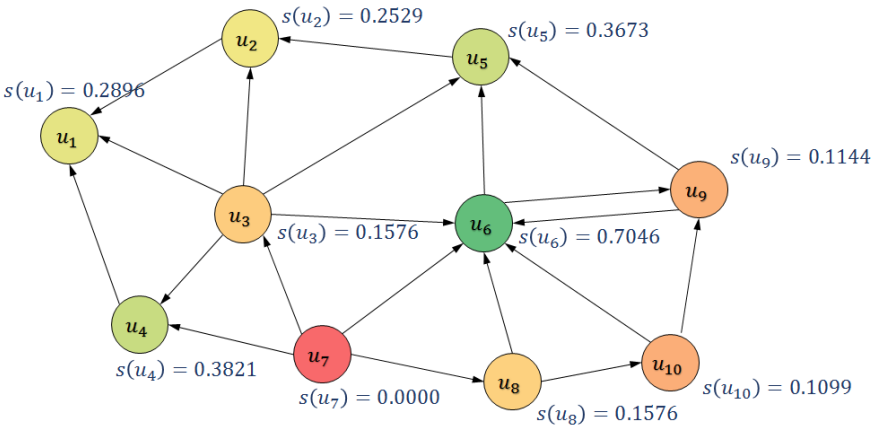


Figura 31. Actuación del algoritmo HITS Authority Recommender en el grafo de entrenamiento de ejemplo.

La tabla 10. presenta las listas recomendaciones que genera el sistema de recomendación haciendo uso de *HITS Authority Recommender* para los distintos usuarios solicitantes de recomendaciones:

u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9	u_{10}
u_6	u_6	u_8	u_6	u_6	u_4	u_5	u_6	u_4	u_4
u_4	u_4	u_9	u_5	u_4	u_1	u_1	u_4	u_5	u_5
u_5	u_5	u_{10}	u_2	u_1	u_2	u_2	u_5	u_1	u_1
u_2	u_3	u_7	u_3	u_3	u_3	u_9	u_1	u_2	u_2
u_3	u_8		u_8	u_8	u_8	u_{10}	u_2	u_3	u_3
u_8	u_9		u_9	u_9	u_{10}		u_3	u_8	u_8
u_9	u_{10}		u_{10}	u_{10}	u_7		u_9	u_7	u_7
u_{10}	u_7		u_7	u_7					
u_7									

Tabla 10. Recomendaciones generadas por el sistema de recomendación utilizando el algoritmo HITS Authority Recommender tomando como datos de entrenamiento el grafo de ejemplo.

Como podemos observar el usuario u_6 es el mejor posicionado en las recomendaciones mientras que el usuario u_7 es el peor posicionado.

Al igual que sucedía con el recomendador no personalizado *Popularity Recommender*, dado que $s_v(u) = s_w(u) = s(u)\ \forall\ u, v, w \in U$, todas las listas de recomendación son el resultado de excluir de la lista de usuarios ordenados por *authority*

$\{u_6, u_4, u_5, u_1, u_2, u_3, u_8, u_9, u_{10}, u_7\}$ el usuario al cual está dirigida la lista de recomendación y los usuarios sobre los que éste ya actúa.

5.1.6.2 HITS Hub

El algoritmo de recomendación no personalizado *HITS Hub* asigna a cada usuario como *score*, independientemente del usuario que solicite las recomendaciones, el *score hub* que se asigna al nodo que representa al usuario susceptible de ser recomendado iterando el algoritmo HITS en el grafo que representa la red social.

Por tanto este algoritmo queda definido por la función *score* s que asigna a cada usuario como *score* su *score hub*:

$$s(u) = hub(u) \tag{17}$$

Donde *hub* es la función presentada en (12).

Si nos fijamos en la segunda mitad de la última fila de la tabla 9 podemos ver cuál es la actuación de este recomendador sobre el grafo de ejemplo (figura 32).

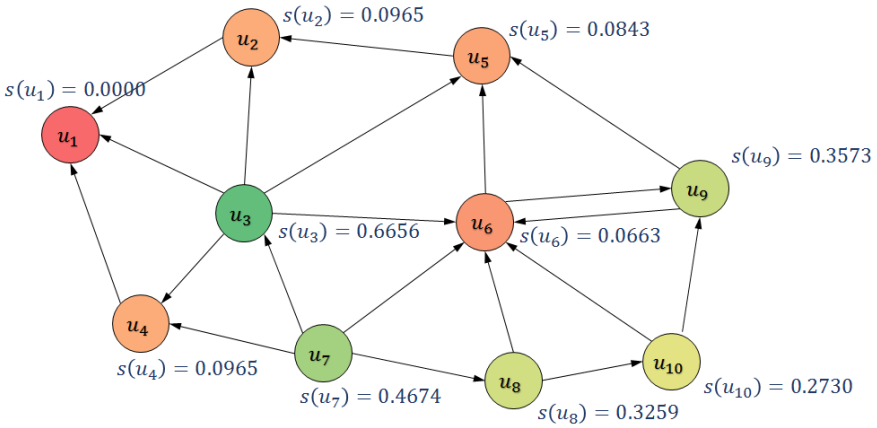


Figura 32. Actuación del algoritmo HITS Hub Recommender en el grafo de entrenamiento de ejemplo.

La tabla 11 presenta las listas recomendaciones que genera el sistema de recomendación haciendo uso de *HITS Hub Recommender* para los distintos usuarios solicitantes de recomendaciones:

u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9	u_{10}
u_3	u_3	u_7	u_3	u_3	u_3	u_9	u_3	u_3	u_3
u_7	u_7	u_9	u_7	u_7	u_7	u_{10}	u_7	u_7	u_7
u_9	u_9	u_8	u_9	u_9	u_8	u_2	u_9	u_8	u_8
u_8	u_8	u_{10}	u_8	u_8	u_{10}	u_5	u_2	u_{10}	u_2
u_{10}	u_{10}		u_{10}	u_{10}	u_2	u_1	u_4	u_2	u_4
u_2	u_4		u_2	u_4	u_4		u_5	u_4	u_5
u_4	u_5		u_5	u_6	u_1		u_1	u_1	u_1
u_5	u_6		u_6	u_1					
u_6									

Tabla 11. Recomendaciones generadas por el sistema de recomendación utilizando el algoritmo HITS Hub Recommender tomando como datos de entrenamiento el grafo de ejemplo.

Como podemos observar el usuario u_3 es el mejor posicionado en las recomendaciones, a pesar de no tener un alto *in-degree* mientras que el usuario u_1 es el peor posicionado.

De nuevo todas las listas de recomendación son el resultado de excluir de la lista de usuarios ordenados por *hub* $\{u_3, u_7, u_9, u_8, u_{10}, u_2, u_4, u_5, u_6, u_1\}$ el usuario al cual está dirigida la lista de recomendación y los usuarios sobre los que éste ya actúa.

5.1.7 Betweenness Centrality

El algoritmo de recomendación no personalizado *Betweenness Centrality Recommender* asigna a cada usuario de la red social un *score* de centralidad basado en la cantidad de caminos más cortos que los contienen. Formalmente queda definido por la siguiente función *score* no personalizado s:

$$s(u) = \sum_{\{v,w \in U : v \neq u \neq w\}} \frac{\sigma_{vw}(u)}{\sigma_{vw}} \tag{18}$$

Donde σ_{uv} representa el número de caminos de distancia mínima del nodo u al nodo v y $\sigma_{uv}(w)$ representa el número de estos caminos que contienen al nodo w.

Podemos ver a modo de ejemplo cómo se comporta este algoritmo con los nodos del grafo de ejemplo. Una observación trivial es que el usuario u_1 va a recibir score cero ya que todos los caminos que lo contienen acaban en él. Lo mismo va a suceder con el usuario u_7 ya que todos los caminos que lo contiene parten de él:

El caso del usuario u_2, que se presenta en la figura 33 es más interesante ya que buena parte de los caminos más cortos que acaban en u_1 contienen a u_2:

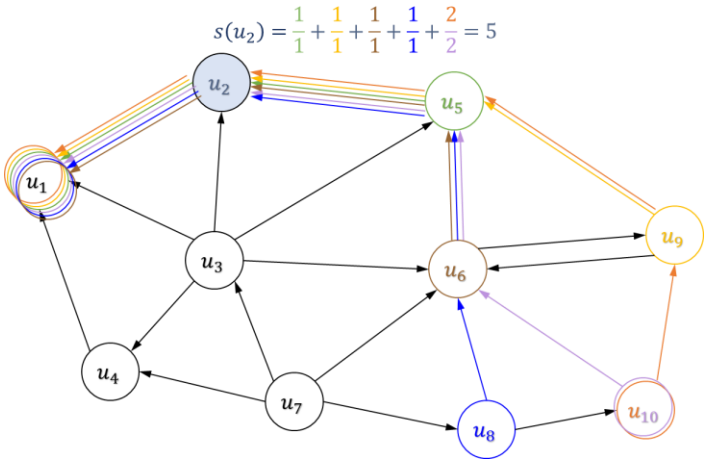


Figura 33. Betweenness Centrality del usuario u_2 en el grafo de ejemplo.

Haciendo esto mismo con todos los usuarios llegamos al siguente reparto de *scores betweenness centrality* que se presenta en la figura 34.

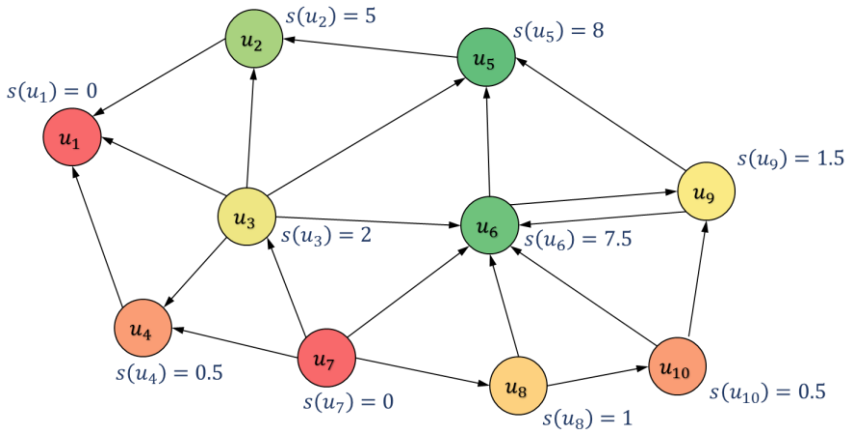


Figura 34. Actuación del algoritmo Betweenness Centrality Recommender en el grafo de entrenamiento de ejemplo.

A continuación presentamos las listas recomendaciones que va a generar el sistema de recomendación haciendo uso de *Betweenness Centrality Recommender* para los distintos usuarios solicitantes de recomendaciones:

u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9	u_{10}
u_5	u_5	u_9	u_5	u_6	u_2	u_5	u_5	u_2	u_5
u_6	u_6	u_8	u_6	u_3	u_3	u_2	u_2	u_3	u_2
u_2	u_3	u_{10}	u_2	u_9	u_8	u_9	u_3	u_8	u_3
u_3	u_9	u_7	u_3	u_8	u_4	u_{10}	u_9	u_4	u_8
u_9	u_8		u_9	u_4	u_{10}	u_1	u_4	u_{10}	u_4
u_8	u_4		u_8	u_{10}	u_1		u_1	u_1	u_1
u_4	u_{10}		u_{10}	u_1	u_7		u_7	u_7	u_7
u_{10}	u_7		u_7	u_7					
u_7									

Tabla 12. Recomendaciones generadas por el sistema de recomendación utilizando el algoritmo Betweenness Centrality Recommender tomando como datos de entrenamiento el grafo de ejemplo.

Como podemos observar el usuario u_5 es el mejor posicionado en las recomendaciones, a pesar de no tener el mayor *in-degree* mientras que los usuarios u_1 y u_7 son los peor posicionados.

Al igual que sucedía con el recomendador no personalizado *Popularity Recommender*, dado que $s_v(u) = s_w(u) = s(u)\ \forall\ u, v, w \in U$, todas las listas de recomendación son el resultado de excluir de la lista de usuarios ordenados por *centrality* $\{u_5, u_6, u_2, u_3, u_9, u_8, u_4, u_{10}, u_1, u_7\}$ el usuario al cual está dirigida la lista de recomendación y los usuarios sobre los que éste ya actúa.

5.2 Algoritmos basados en el contenido

Los algoritmos basados en contenido que hemos implementado se apoyan en el procesamiento offline de *tweets* llevado a cabo por los programas *Indexer*, *InteractionsMapBuilder* y *CentroidBuilder* presentados en los apartados 4.7, 0 y 4.9 respectivamente.

Al igual que en la presentación de los algoritmos basados en la topología de red social, cada recomendador va a ser descrito formalmente con una función generadora de score s_v de acuerdo con la ecuación (5) ya que los algoritmos basados en contenido son de naturaleza personalizada.

5.2.1 Rocchio

El algoritmo Rocchio (Rocchio, 1971) se basa en la similitud entre el usuario susceptible de recibir recomendaciones y su similitud con los distintos ítems susceptibles de ser recomendados.

En nuestro caso particular, tras la ejecución de los tres programas *Indexer*, *InteractionsMapBuilder* y *CentroidBuilder* contamos con representaciones vectoriales de todos los usuarios del conjunto de datos de entrenamiento que nos permiten emplear como medida de similitud el coseno de dichos vectores. De este modo, nuestra implementación de este algoritmo queda definida por la función *score* personalizada s_v definida como:

$$s_v(u) = \cos(\vec{u}, \vec{v}) = \frac{\vec{u} \cdot \vec{v}}{|\vec{u}||\vec{v}|} \tag{19}$$

Donde $\vec{u}$ representa el *centroide* del usuario u susceptible de ser recomendado al usuario v y $\vec{v}$ representa el *centroide* del usuario v susceptible de recibir recomendaciones.

En función del algoritmo de decisión de pesos de los *tweets* para la construcción de los *centroides* de usuario durante la ejecución del programa *InteractionsMapBuilder* hemos definido dos variantes del algoritmo que difieren en los pesos de cada coordenada en los *centroides*. A continuación presentamos las dos variantes cuyo algoritmo de ponderación de *tweets* para la construcción de los *centroides* de usuarios queda definido por la función de peso personalizada w_u:

$$w_u : T \rightarrow [0, +\infty) \tag{20}$$

Donde T representa el conjunto de todos los tweets del conjunto de datos de entrenamiento y u es el usuario para la construcción de cuyo *centroide* se están asignado pesos a cada *tweet* del conjunto de datos.

5.2.1.1 Rocchio constante para tweets escritos por el usuario

En esta variante, a la hora de asignar pesos a los *tweets* del conjunto de datos para construir el *centroide* descriptivo de cada usuario, se valoran por igual todos los *tweets* publicados originalmente por el usuario (es decir, aquellos publicados por el usuario que no sean *retweets*). El resto de los tweets del conjunto de datos no serán considerados para la construcción del centroide de usuario.

Este algoritmo de asignación de pesos queda definido formalmente por la función de asignación de pesos w_u definida como:

$$w_u(t) = \begin{cases} 1 & \text{si } t \in T_u \\ 0 & \text{si } t \notin T_u \end{cases} \tag{21}$$

Donde T_u es el subconjunto de los *tweets* del conjunto de datos de entrenamiento cuyo autor es el usuario u que no son *retweets* de *tweets* previamente publicados por otro usuario.

5.2.1.2 Rocchio dependiente del tiempo para tweets escritos por el usuario

En esta variante a la hora de asignar pesos a los *tweets* del conjunto de datos para construir el *centroide* descriptivo de cada usuario se penalizan los *tweets* menos recientes publicados originalmente por el usuario y se da más peso a los *tweets* más recientes.

En particular hemos optado adaptar la ponderación del peso a una función lineal del orden temporal de los *tweets* publicados por el usuario, de este modo, este algoritmo de asignación de pesos queda definido formalmente por la función de asignación de pesos w_u definida como:

$$w_u(t) = \begin{cases} \dfrac{|T_u| - n_{T_u}(t) + 1}{|T_u|} \text{ si } t \in T_u \\ 0 \text{ si } t \notin T_u \end{cases} \quad (22)$$

Donde T_u, al igual que en la ecuación (21), es el subconjunto de los *tweets* del conjunto de datos de entrenamiento cuyo autor es el usuario u que no son *retweets* de *tweets* previamente publicados por otro usuario. Y $n_{T_u}(t)$ es la posición del *tweet* t en la ordenación de los *tweets* del subconjunto T_u por fecha en orden de más reciente a más antigua.

5.3 Sistema de hibridación de algoritmos

Como hemos podido observar, parte de los algoritmos de recomendación presentados son personalizados, otros no lo son y todos se basan en ideas no necesariamente incompatibles a la hora de establecer rankings de usuarios recomendados: es natural plantearse la utilización conjunta de dos o más de los algoritmos de recomendación para obtener nuevas recomendaciones más flexibles.

Para utilizar conjuntamente distintos algoritmos encontramos dos sub-problemas que debemos resolver: El primero de ellos concierne a la diferencia en las distribuciones de los *score* generados por los distintos algoritmos de recomendación a la hora de generar los rankings y el segundo concierne a la generación del ranking final a partir de los *scores* normalizados.

Esto divide nuestro sistema de hibridación de algoritmos en dos fases en cada una de las cuales hemos implementado una solución clásica al respectivo sub-problema planteado. Por último abordamos la integración del sistema de hibridación de algoritmos como algoritmo híbrido de recomendación de forma que podamos realizar experimentos de forma automatizada.

5.3.1 Normalización de scores: rank-sim

Cada algoritmo implementado asigna un *score* a cada par usuario-ítem que permite generar un ranking de ítems que conforma la lista ordenada de ítems recomendados al usuario.

Los *score* generados por los distintos algoritmos tienen distribuciones e incluso rangos de valores distintos y no dejan de ser un instrumento de ordenación para generar el ranking que es el output final de todo recomendador. Por este motivo hemos optado por extraer un *score* normalizado del propio ranking generado por cada recomendador involucrado en el proceso de recomendación híbrida de forma que los nuevos *score* sean idénticamente distribuidos y pertenezcan al mismo intervalo permitiendo así su utilización conjunta en la generación del ranking final.

Para hacer esto hemos optado por el método lineal *rank-sim* que queda definido con la siguiente ecuación de normalización de *score* personalizada $\overline{s_v}$:

$$\overline{s_v}(u) = \frac{|R_v| - rank_{R_v}(u) + 1}{|R_v|} \tag{23}$$

Donde u es un usuario recomendado al usuario v susceptible de recibir recomendaciones, R_v es la lista ordenada de usuarios recomendados al usuario v y $rank_{R_v}(u)$ es la posición del usuario u en la lista ordenada por *score* de usuarios recomendados al usuario v.

Nótese que utilizando un *tweet* t en lugar de un usuario u, la función de normalización de *score* sigue funcionando. Es decir, hasta este punto el sistema de hibridación de algoritmos es compatible no sólo con algoritmos de recomendación de usuarios sino también con algoritmos de recomendación de *tweets*. Es importante también tener en cuenta que antes de comenzar a obtener *scores* normalizados es necesario haber asignado un *score* a cada ítem susceptible de ser recomendado para poder conocer la posición en la lista de recomendaciones del ítem cuyo score queremos normalizar.

5.3.2 Generación del ranking final: combinación lineal ponderada

Una vez contamos con *scores* normalizados para cada recomendador podemos generar el ranking final que representa la lista ordenada de recomendaciones generada de forma conjunta por los distintos algoritmos de recomendación involucrados en la hibridación.

Es razonable tener la capacidad de modificar el peso de los distintos algoritmos en la generación de recomendaciones ya que esto nos permite mucha flexibilidad a la hora de buscar la combinación óptima de algoritmos de recomendación para generar recomendaciones. Por esto motivo hemos optado por seguir la metodología de combinación lineal ponderada que a partir de los *score* normalizados tal y como se describe en el sub-apartado anterior, calcula un nuevo *score* s_v que es el que sirve para generar el ranking final:

$$s_v(u) = \sum_i w_i \bar{s}_{v_i}(u) \tag{24}$$

Donde el iterador i recorre todos los algoritmos involucrados en la recomendación híbrida, w_i representa para cada algoritmo i su peso en la generación de la recomendación híbrida y $\bar{s}_{v_i}(u)$ representa el *score* normalizado conforme a la ecuación (23) asignado por el algoritmo i al usuario u susceptible de ser recomendado al usuario v.

De nuevo nótese que utilizando un *tweet* t en lugar de un usuario u, la función de combinación de *score* sigue funcionando. Es decir, hasta este punto el sistema de hibridación de algoritmos es compatible no sólo con algoritmos de recomendación de usuarios sino también con algoritmos de recomendación de *tweets*.

5.3.3 Integración como algoritmo híbrido de recomendación

Para permitir experimentar de forma rápida y flexible combinando distintos algoritmos de recomendación, hemos optado por integrar el sistema de hibridación de algoritmos como algoritmo híbrido de recomendación implementando la interfaz genérica de recomendación mencionada en el apartado 4.10.

Esta interfaz contiene un método de preparación de los recomendadores para cada usuario en el cual el sistema de hibridación calcula ranksim para cada uno de los algoritmos de recomendación involucrados. Estos algoritmos a su vez se configuran mediante el método de configuración, que en el caso de la implementación del algoritmo híbrido de recomendación, admite los nombres, pesos y parámetros de configuración de cada algoritmo de recomendación que se quiera utilizar en cada experimento.

De este modo disponemos finalmente de un sistema de hibridación de algoritmos de uso flexible que nos permite realizar múltiples experimentos de combinación de algoritmos de recomendación.

6. Experimentación

Presentamos aquí los resultados experimentales de evaluar los algoritmos de recomendación implementados con distintas métricas de precisión, novedad y diversidad. Comenzamos describiendo el diseño de los experimentos que se han realizado detallando las configuraciones de los conjuntos de datos de entrenamiento y test empleadas para la experimentación, los algoritmos de recomendación utilizados para generar recomendaciones experimentales así como sus parámetros de configuración y presentando las métricas que se han utilizado para evaluar los distintos algoritmos de recomendación. Por último presentamos los resultados agregados de los experimentos realizados procediendo a un análisis de los mismos.

6.1 Configuración experimental

Los experimentos realizados se dividen en dos grandes bloques en función de la naturaleza de los datos empleados para generar recomendaciones. Siguiendo la nomenclatura establecida en el apartado 4.5, por una parte hemos realizado experimentos con el `TwitterData` asociado al grafo de interacciones sociales de tipo *follow* y por otra parte hemos realizado experimentos con el `TimeAwareTwitterData` asociado al grafo de interacciones sociales *mention / reply / retweet*.

En el primero de los dos conjuntos de datos, en el que no se incluyen *tweets* ni información temporal, hemos realizado una partición aleatoria mientras que en el segundo conjunto de datos, que cuenta con *tweets* e información temporal, hemos realizado una partición cronológica. En ambos casos se ha configurado el programa *splitter* para utilizar el 60% de los datos en el conjunto de datos de entrenamiento y el 40% de los datos en el conjunto de test. En el caso particular del `TimeAwareTwitterData`, dada la redundancia de interacciones, utilizando el 60% de los *tweets* se recogen en el grafo del conjunto de entrenamiento el 66.56% de las interacciones del grafo inicial quedando el 33.34% de las interacciones en el grafo de test.

En ambos conjuntos de datos experimentamos con todos los algoritmos de recomendación basados en la topología de red social presentados en el apartado 5.1 y en particular combinamos, haciendo uso del sistema de hibridación de algoritmos, los algoritmos MaxFOAF y Popularity.

En el caso particular del conjunto de datos `TimeAwareTwitterData`, ya que cuenta con tweets asociados, experimentamos con las dos variantes del algoritmo Rocchio, basado en contenido, presentado en el apartado 5.2 y ampliamos los experimentos de hibridación combinando sendas variantes de dicho algoritmo con los algoritmos MaxFOAF, Popularity y PageRank.

Dado que los algoritmos PageRank y Personalized PageRank cuentan con un parámetro configurable (el factor de teleportación) que admite valores en el intervalo semiabierto [0,1], cada vez que realizamos experimentos con cualquiera de estos dos algoritmos de recomendación realizamos un barrido de este parámetro de configuración tomando valores de 0 a 1 con saltos de 0.1 entre cada dos valores, es decir, experimentamos con factores de teleportación en el conjunto {0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1}.

En los experimentos en los que combinamos dos algoritmos de recomendación realizamos igualmente un barrido del peso del primer algoritmo en el intervalo cerrado [0,1] con saltos de 0.1 siendo el peso del segundo algoritmo 1 menos el peso del primero, es decir, experimentamos con pesos de cada algoritmo en el conjunto {0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1}.

En la tabla 13 presentamos de forma agregada todos los algoritmos de recomendación empleados para generar y evaluar recomendaciones y sus parámetros de configuración:

	Follow	***Mention / Reply / Retweet***
Experimentos individuales	Popularity MaxFOAF Random PageRank * PersonalizedPageRank * HITS Authority HITS Hub Betweenness Centrality	Popularity MaxFOAF Random PageRank * PersonalizedPageRank * HITS Authority HITS Hub Betweenness Centrality Rocchio CFSW*** Rocchio LDFSW***
Experimentos híbridos **	MaxFOAF – Popularity	MaxFOAF – Popularity Popularity - Rocchio CFSW Popularity - Rocchio LDFSW MaxFOAF – Rocchio CFSW MaxFOAF – Rocchio LDFSW PageRank * - Rocchio CFSW PageRank * - Rocchio LDFSW

* Con factor de teleportación tomando todos los valores del conjunto {0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1}.

** Con pesos de los dos algoritmos de recomendación tomando todos los valores del conjunto {0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1} y sumando 1 en todo experimento.

*** CFSW abrevia Costant For Self Written, LDFSW abrevia LDependant For Self Written.

Tabla 13. Algoritmos de recomendación utilizados en los experimentos y valores tomados por sus parámetros de configuración.

En lo que respecta a las métricas con las que se evalúan las recomendaciones generadas por los algoritmos de recomendación configurados conforme a la tabla anterior, en todo experimento hemos utilizado las métricas de precisión, novedad y diversidad que se describen en detalle en los siguientes sub-apartados, pero antes conviene presentar dos aspectos comunes a todas ellas:

- Todas las métricas están orientadas a la evaluación de la lista de recomendaciones generada para un usuario, no obstante, es natural a la hora de evaluar un algoritmo de recomendación promediar sobre todos los usuarios cuyas recomendaciones han sido evaluadas.
- Todas las métricas admiten como parámetro de configuración el *cutoff* (k) que determina la cantidad máxima de ítems recomendados que se toman en consideración a la hora de evaluar las recomendaciones generadas para el usuario. En el caso de las métricas de precisión, tomar *cutoffs* bajos es natural y aporta una buena estimación de la calidad efectiva de los recomendadores ya que el usuario la mayor parte de las veces se limita a la observación de los primeros ítems aparecidos en el ranking.

6.1.1 Métricas de precesión

Para valorar la precisión de las recomendaciones hemos recurrido a la herramienta externa *trec_eval*. En particular hacemos uso de su implementación de las métricas clásicas orientadas a relevancia *P* (precisión) y *nDCG*. Describimos a continuación el funcionamiento de las dos métricas y presentamos su formulación matemática:

6.1.1.1 Precisión

La precisión (Baeza & Ribeiro, 1999) representa la aproximación más natural a la estimación de la tasa de resultados relevantes de un algoritmo de recomendación o de un sistema de recuperación de información. Se basa directamente en la proporción que representan los ítems relevantes dentro del conjunto de los ítems recomendados:

$$P@k = \frac{|R \cap D@k|}{k} \tag{25}$$

Donde R son los ítems relevantes para el usuario cuyas recomendaciones se están evaluando. Esta lista de ítems los lee la herramienta *trec_eval* del fichero de juicios de relevancia. $D@k$ es el subconjunto de los primeros k ítems que el algoritmo de recomendación que estamos evaluando ha recomendado al usuario.

6.1.1.2 *nDCG*

La métrica *nDCG Normalized Discounted Cumulated Gain* (Jarvelin & Kekalainen, 2001) va un paso más allá que la métrica presentada en el sub-apartado anterior dando importancia a la posición de los ítems recomendados dentro del ranking generado por el algoritmo de recomendación: es más valioso, con la misma precisión, un algoritmo que sitúa los ítems relevantes en las primeras posiciones que un algoritmo que los sitúa más abajo, incluso cuando en total el número de relevantes devueltos fuese el mismo. Antes de presentar la formulación final de la métrica de precisión *nDCG* conviene entender el desarrollo conceptual que lleva a la misma:

Partimos del concepto de CG (*Cumulated Gain)* dado un *cutoff* k en el ranking de ítems generado por un recomendador:

$$CG@k = \sum_{j=1}^{|D@k|} rel_j \tag{26}$$

Donde rel_j representa la relevancia del ítem en la posición j de ranking de ítems recomendados. Es decir, la GC representa la relevancia acumulada en una lista de recomendaciones. Dado que en nuestro caso los juicios de relevancia generados por el programa *Splitter* son siempre binarios (1 para usuarios con los que se observa en el grafo de test que el usuario actúa, 0 para el resto de usuarios), una expresión equivalente simplificada a nuestro escenario de la ganancia acumulada es la siguiente:

$$CG@k = \sum_{R \cap D@k} 1 = |R \cap D@k| \tag{27}$$

Donde $D@k$ es, al igual que en la ecuación (25) el subconjunto de los primeros como máximo k ítems que el algoritmo de recomendación que estamos evaluando ha recomendado al usuario cuyas recomendaciones estamos evaluando.

Para valorar la posición de cada ítem relevante en el ranking, aparece el concepto de descuento: vamos a modificar ligeramente la expresión CG para reducir la contribución de los ítems relevantes peor posicionados a la ganancia acumulada llegando así al concepto DCG (*Discounted Cumulated Gain)*:

$$DCG@k = \sum_{j=1}^{|D@k|} \frac{rel_j}{\log_2(j+1)} \tag{28}$$

De nuevo podemos presentar una expresión equivalente adaptada a nuestro escenario binario:

$$DCG@k = \sum_{u \in R \cap D@k} \frac{1}{\log_2(rank(u)+1)} \tag{29}$$

Donde u representa un usuario o ítem perteneciente a la lista de recomendaciones $D@k$ y al conjunto de usuarios relevantes R y $rank(u)$ es su posición en la lista de recomendaciones $D@k$.

Dado que no todos los usuarios tienen los mismos juicios de relevancia pudiendo variar la longitud de los mismos y en el caso general incluso los *ratings* de relevancia, es natural proceder a una normalización de DCG de forma que podamos comparar equitativamente la métrica para distintos usuarios y algoritmos de recomendación. Una forma natural es trasladar DCG a la escala $[0,1]$ dividiendo entre el máximo DCG posible que se denomina $IDCG$ (*Ideal Discounted Cumulated Gain*):

$$IDCG@k = \max_{D \in \sigma(R)} \sum_{j=1}^{\min\{k,|R|\}} \frac{rel_j}{\log_2(j+1)} \tag{30}$$

Donde rel_j, al igual que en la ecuación (26) representa la relevancia del ítem en la posición j de ranking de ítems recomendados D. De nuevo podemos presentar una expresión equivalente adaptada a nuestro escenario binario:

$$IDCG@k = \sum_{j=1}^{\min\{k,|R|\}} \frac{1}{\log_2(j+1)} \tag{31}$$

De esta forma llegamos a la expresión final de la métrica $nDCG$:

$$nDCG@k = \frac{DCG@k}{IDCG@k} = \frac{\sum_{j=1}^{|D@k|} \frac{rel_j}{\log_2(j+1)}}{\max_{D \in \sigma(R)} \sum_{j=1}^{\min\{k,|R|\}} \frac{rel_j}{\log_2(j+1)}} \tag{32}$$

Que en el caso particular de juicios de relevancia binarios queda simplificada a:

$$nDCG@k = \frac{DCG@k}{IDCG@k} = \frac{\sum_{u \in R \cap D@k} \frac{1}{\log_2(rank(u)+1)}}{\sum_{j=1}^{\min\{k,|R|\}} \frac{1}{\log_2(j+1)}} \tag{33}$$

En definitiva, lo que tenemos es una métrica normalizada en el intervalo [0,1] que valora no sólo la cantidad de ítems relevantes devueltos sino también su posicionamiento en la lista de recomendaciones.

6.1.2 Métricas de novedad y diversidad

Tradicionalmente las métricas de precisión han marcado la línea de referencia a la hora de comparar algoritmos de recomendación pero en los últimos tiempos se empieza a investigar en direcciones alternativas buscando la mejora de propiedades de los algoritmos de recomendación tales como su novedad y su diversidad. En el contexto de este trabajo de fin de grado nos hemos limitado a experimentar con dos métricas sencillas que presentamos en los siguientes sub-apartados pero aprovechando el conjunto de datos y la relativa novedad de esta línea de estudio hemos trabajado en paralelo experimentando con distintas métricas de novedad y diversidad así como algoritmos de recomendación avaros dando lugar a la publicación del póster (Alhambra, et al., 2014).

6.1.2.1 Novedad

La métrica de novedad *EPC* (*Expected Popularity Complement)* se basa en la idea opuesta a la que define el algoritmo de recomendación *Popularity* presentado en el apartado 5.1.1: una lista de recomendaciones se evaluará con mayor EPC cuanto menor sea la popularidad de los usuarios que contiene. Procedamos a la descripción formal de esta métrica comenzando por definir la popularidad de un usuario en el rango [0,1] dividiendo el número de seguidores del usuario entre el número total de usuarios en el conjunto de datos:

$$p(u) = \frac{\sum_{\{v \in U: v \neq u\}} \delta_u(v)}{|U| - 1} \quad (34)$$

Donde U representa el del conjunto de todos los usuarios del datase. Esta fórmula entendida en términos probabilísticos se puede interpretar como la probabilidad de que un usuario elegido al azar en el conjunto de datos siga al usuario u y consecuentemente el usuario u ya no sea novedoso para ese usuario.

Es natural consecuentemente definir como novedad de un usuario u la probabilidad complementaria:

$$nov(u) = 1 - p(u) = 1 - \frac{\sum_{\{v \in U: v \neq u\}} \delta_u(v)}{|U| - 1} \quad (35)$$

A partir de esta noción de novedad podemos proceder a formular la métrica EPC:

$$EPC@k = \frac{1}{|D@k|} \sum_{u \in D@k} nov(u) \quad (36)$$

La métrica puede interpretarse como la probabilidad a priori de que el usuario objetivo conozca uno de los usuarios recomendados tomado al azar.

Como podremos observar en los resultados experimentales, el recomendador *Popularity* registrará valores entre los más bajos de EPC. Esto es natural dada la formulación complementaria de la métrica y el algoritmo de recomendación.

6.1.2.2 Diversidad

La métrica de diversidad ***ILD*** (*Intra List Dissimilarity*) se basa en la diferencia entre los ítems dentro de una misma lista de ítems recomendados.

Antes de proceder a formular la métrica es importante tener en cuenta que existen diversos enfoques de similitud entre usuarios. En nuestro caso particular, dado que no siempre disponemos de tweets asociada a los mismos, hemos optado por la noción de similitud basada en el vecindario de los usuarios:

$$sim(u, v) = \frac{|\{w \in U: \delta_w(u) = 1\} \cap \{w \in U: \delta_w(v) = 1\}|}{|\{w \in U: \delta_w(u) = 1\} \cup \{w \in U: \delta_w(v) = 1\}|} \quad (37)$$

Donde la función δ_u es la definida previamente en la ecuación (7).

Nótese que esta noción de similitud, naturalmente es simétrica ($sim(u, v) = sim(v, u)$) y nos devuelve un valor de similitud de usuarios en el intervalo $[0,1]$ proporcional al ratio entre el número de usuarios que siguen ambos usuarios y el número total de usuarios que siguen entre los dos usuarios. En definitiva, se trata de la similitud de Jaccard entre los respectivos conjuntos de usuarios seguidos por los dos usuarios.

Al igual que en el caso de la noción de novedad, complementamos para cuantificar la diferencia en función de la similitud:

$$dif(u, v) = 1 - sim(u, v) \quad (38)$$

A partir de esta noción de diversidad podemos proceder a formular la métrica ILD:

$$ILD@k = \frac{1}{k(k-1)} \sum_{u \in D@k} \sum_{v \in D@k \setminus \{u\}} dif(u, v) \quad (39)$$

6.2 Resultados

A continuación procedemos a presentar de forma agregada los resultados de evaluar los experimentos de recomendación presentados en la tabla 13 haciendo uso de las métricas presentadas en el apartado anterior. En particular vamos a mostrar en este apartado los resultados de la evaluación de las métricas para las configuraciones de los algoritmos que maximizan la evaluación de la métrica de precisión con *cutoff* 5. Para conocer más detalles acerca de los resultados experimentales resultantes de evaluar las distintas configuraciones de los algoritmos de recomendación que admiten parámetros de configuración, se remite al lector al anexo 4.

6.2.1 Resultados en el conjunto de datos *Follow*

La tabla 14 muestra el resumen de los resultados obtenidos en los experimentos realizados en el conjunto de datos *follow*. Para apreciar mejor las diferencias cuantitativas entre los distintos algoritmos, la figura 35 muestra un resumen de estos datos en forma de diagrama de barras.

6.2.1.1 Observaciones acerca de los resultados de precisión y nDCG

Se observa que la combinación de los algoritmos *MaxFOAF* y *Popularity* tomando pesos 0,6 y 0,4 respectivamente, obtiene mejores resultados con las métricas precisión y *nDCG* que los dos algoritmos por separado. Siguen a esta combinación estos dos mismos algoritmos por separado siendo el segundo considerablemente menos preciso que el primero.

El algoritmo *HITS Authority* presenta resultados de precisión y *nDCG* muy similares a los de *Popularity* y es seguido de lejos por el algoritmo *Betwenness Centrality* que además empeora sus resultados de precisión y *nDCG* a medida que aumenta el *cutoff* más rápido que ningún otro algoritmo.

Llama la atención el algoritmo *PageRank* personalizado, que a pesar de presentar resultados de precisión y *nDCG* muy discretos para *cutoffs* bajos, al contrario de lo que sucede con los algoritmos contra los que compite en estas dos métricas, mejora sus resultados a medida que aumenta el *cutoff* llegando a ser el tercer mejor algoritmo en precisión y *nDCG* para el *cutoff* 50. El algoritmo *PageRank* no personalizado por contra, a pesar de presentar resultados similares con *cutoffs* bajos, resulta claramente peor en términos de precisión y *nDCG* que la versión personalizada. Esto prueba que en este escenario personalizar el algoritmo *PageRank* supone una mejora tangible en la recomendación de usuarios.

Por último queda el algoritmo *HITS Hub*, que presenta resultados de precisión y *nDCG* casi tan bajos como los presentados por el algoritmo de línea base *Random*. Esto era de esperar dada la naturaleza de este algoritmo: los usuarios que mejor *score* que

recomienda este algoritmo son aquellos que más usuarios relevantes siguen en términos de autoridad y no los usuarios más relevantes.

	P@5	P@10	P@15	P@20	P@30	P@50	nDCG@5	nDCG@10	nDCG@15	nDCG@20	nDCG@30	nDCG@50	EPC@50	ILD@50
Híbrido MaxFOAF 0.6 + Popularity 0.4	0.1882	0.1676	0.1554	0.1463	0.1328	0.1144	0.2003	0.1903	0.1888	0.1907	0.1996	0.2223	0.9919	0.9750
MaxFOAF	0.1709	0.1525	0.1411	0.1329	0.1215	0.1059	0.1824	0.1728	0.1703	0.1713	0.1790	0.2002	0.9964	0.9776
Popularity	0.0635	0.0561	0.0506	0.0471	0.0428	0.0377	0.0661	0.0619	0.0594	0.0588	0.0614	0.0685	0.9608	0.9933
HITSAuthority	0.0627	0.0512	0.0465	0.0435	0.0400	0.0338	0.0655	0.0577	0.0550	0.0539	0.0547	0.0574	0.9648	0.9840
BetwennessCentrality	0.0343	0.0232	0.0218	0.0186	0.0199	0.0200	0.0402	0.0315	0.0308	0.0292	0.0325	0.0397	0.9792	0.9953
PersonalizedPageRank 0.1	0.0279	0.0285	0.0300	0.0320	0.0365	0.0425	0.0301	0.0333	0.0385	0.0446	0.0584	0.0862	0.9823	0.9885
PageRank 0.1	0.0232	0.0228	0.0267	0.0266	0.0255	0.0229	0.0283	0.0277	0.0318	0.0338	0.0377	0.0450	0.9762	0.9959
HITSHub	0.0125	0.0087	0.0081	0.0066	0.0080	0.0075	0.0164	0.0124	0.0113	0.0100	0.0111	0.0118	0.9922	0.9847
Random	0.0024	0.0026	0.0025	0.0026	0.0026	0.0025	0.0026	0.0026	0.0027	0.0029	0.0032	0.0039	0.9974	0.9989

Tabla 14. Tabla resumen de resultados experimentales en el conjunto de datos Follow.

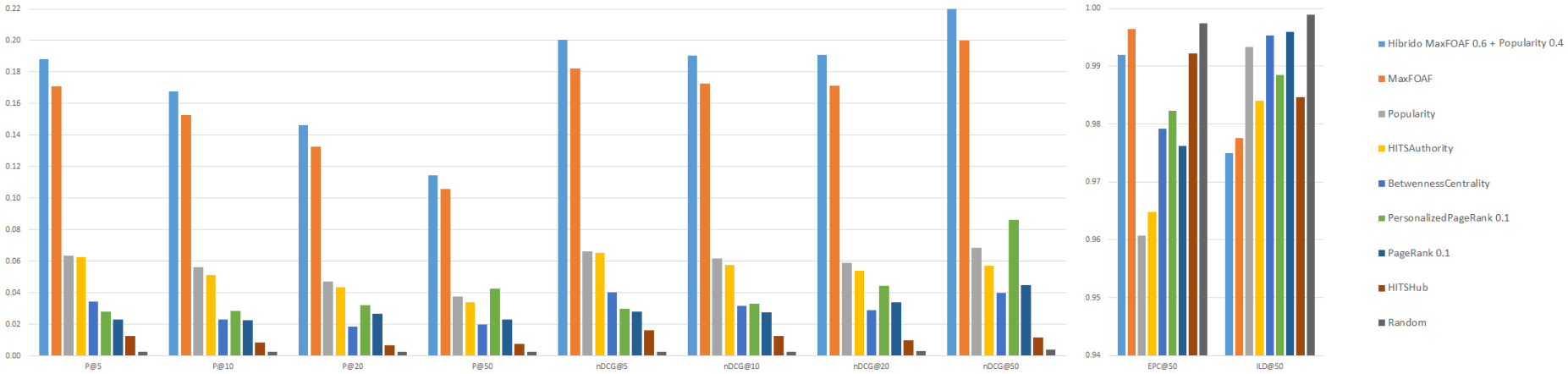


Figura 35. Gráfica resumen de resultado experimentales en el conjunto de datos Follow.

6.2.1.2 Observaciones acerca de los resultados de novedad y diversidad

En lo que respecta a novedad y diversidad, naturalmente, el algoritmo línea base para precisión y *nDCG Random* es el que mejores resultados obtiene de forma indiscutible. Esto se entiende si tenemos en cuenta que no se basa en ningún criterio estructurado para recomendar usuarios y consecuentemente los usuarios recomendados sólo pueden parecerse entre sí por casualidad (lo cual aumenta la diversidad) y sólo pueden resultar familiares al usuario que recibe las recomendaciones por casualidad (lo cual aumenta la novedad).

Llama nuestra atención el alto valor de novedad que registra el algoritmo *MaxFOAF* tanto en solitario como, de forma más atenuada, en su combinación con el algoritmo *Popularity*. Este resultado podría llegar a alarmarnos si tenemos en cuenta que el algoritmo *MaxFOAF* recomienda de algún modo a los usuarios más populares (y consecuentemente menos novedosos) entre los usuarios del subconjunto de la red social formado por los usuarios seguidos por los usuarios que sigue el usuario. No obstante, al buscar usuarios populares dentro de ese subconjunto del cual, los usuarios que más veces quedan expulsados son precisamente los más populares (es decir, aquellos que son seguidos por más usuarios) el algoritmo *MaxFOAF* consigue mantener niveles altos de novedad en este conjunto de datos. En el caso de la métrica de diversidad, no obstante, el algoritmo *MaxFOAF*, tanto en solitario como en combinación con el algoritmo *Popularity* los valores de diversidad son los más bajos registrados entre todos los algoritmos de recomendación evaluados.

El algoritmo que sí que minimiza la novedad en las recomendaciones es naturalmente *Popularity*. Este resultado es natural si recordamos que las definiciones de popularidad y novedad de un usuario que hemos tomado en consideración para definir el algoritmo de popularidad y la métrica de novedad son complementarias. En el caso de la métrica de diversidad, el algoritmo *Popularity* sí que presenta valores relativamente altos en comparación con los presentados por los algoritmos que compiten con él en precisión. Esto se entiende en el sentido de que los usuarios más populares no necesariamente tienen que ser similares entre sí.

6.2.2 Resultados en el conjunto de datos *Mention / Reply / Retweet*

La tabla muestra el resumen de los resultados obtenidos en los experimentos realizados en el conjunto de datos *mention / reply / retweet*. Para apreciar mejor las diferencias cuantitativas entre los distintos algoritmos, la figura muestra un resumen de estos datos en forma de diagrama de barras.

	P@5	P@10	P@15	P@20	P@30	P@50	nDCG@5	nDCG@10	nDCG@15	nDCG@20	nDCG@30	nDCG@50	EPC@50	ILD@50
HITSAuthority	0.0447	0.0342	0.0277	0.0237	0.0202	0.0165	0.0540	0.0516	0.0514	0.0523	0.0577	0.0659	0.9815	0.9949
Híbrido MaxFOAF 0.1 + Popularity 0.9	0.0442	0.0374	0.0338	0.0303	0.0252	0.0194	0.0527	0.0547	0.0594	0.0631	0.0696	0.0789	0.9829	0.9975
Popularity	0.0414	0.0321	0.0281	0.0252	0.0205	0.0160	0.0538	0.0525	0.0551	0.0582	0.0623	0.0704	0.9821	0.9982
PageRank_0.3	0.0402	0.0304	0.0252	0.0215	0.0178	0.0143	0.0543	0.0532	0.0545	0.0561	0.0610	0.0688	0.9840	0.9981
Hybrid MaxFOAF 0.9 + Rocchio LDFSW 0.1	0.0246	0.0250	0.0242	0.0234	0.0214	0.0178	0.0286	0.0350	0.0409	0.0462	0.0550	0.0666	0.9969	0.9959
Híbrido MaxFOAF 0.9 + Rocchio CFSW 0.1	0.0245	0.0250	0.0240	0.0233	0.0213	0.0176	0.0283	0.0348	0.0406	0.0460	0.0544	0.0659	0.9967	0.9959
MaxFOAF	0.0222	0.0233	0.0228	0.0218	0.0201	0.0166	0.0256	0.0323	0.0382	0.0428	0.0513	0.0619	0.9978	0.9957
BetwennessCentrality	0.0202	0.0154	0.0149	0.0123	0.0092	0.0076	0.0286	0.0263	0.0283	0.0288	0.0298	0.0339	0.9915	0.9981
PersonalizedPageRank_0.1	0.0179	0.0225	0.0244	0.0250	0.0253	0.0235	0.0174	0.0260	0.0334	0.0408	0.0543	0.0740	0.9939	0.9945
Rocchio LDFSW	0.0173	0.0116	0.0090	0.0077	0.0062	0.0047	0.0242	0.0220	0.0216	0.0221	0.0234	0.0259	0.9961	0.9996
Rocchio CFSW	0.0166	0.0108	0.0084	0.0072	0.0057	0.0044	0.0241	0.0213	0.0209	0.0213	0.0225	0.0249	0.9958	0.9997
Random	0.0013	0.0015	0.0013	0.0012	0.0013	0.0013	0.0013	0.0018	0.0020	0.0022	0.0029	0.0042	0.9986	0.9995
HITSHub	0.0007	0.0008	0.0024	0.0020	0.0015	0.0014	0.0005	0.0008	0.0025	0.0026	0.0027	0.0036	0.9984	0.9987

Tabla 15. Tabla resumen de resultados experimentales en el conjunto de datos Mention / Reply / Retweet.

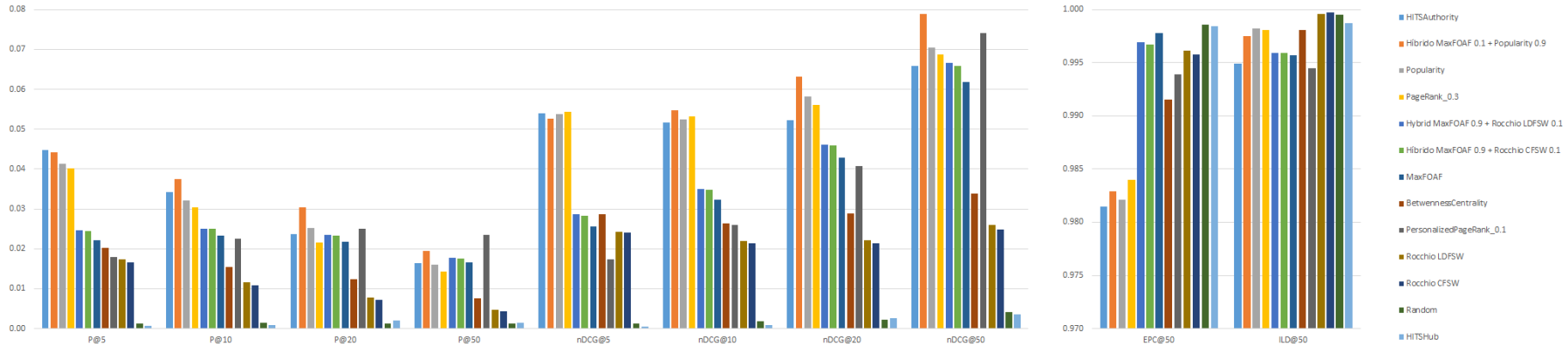


Figura 36. Gráfica resumen de resultado experimentales en el conjunto de datos Mention /Reply / Retweet.

6.2.2.1 Observaciones acerca de los resultados de precisión y nDCG

Se observa en los resultados presentados que esta vez el algoritmo *HITS Authority* se presenta como el más efectivo en términos de precisión y *nDCG* para *cutoffs* bajos. No obstante, el híbrido *MaxFOAF + Popularity* adquiere importancia a medida que aumenta el *cutoff*.

Es llamativo el caso de los algoritmos *PageRank* en sus dos configuraciones en sus variantes personalizada y no personalizada. Por una parte la variante no personalizada presenta mejores evaluaciones de precisión y *nDCG* que la variante no personalizada para *cutoffs* bajos, no obstante, a medida que aumenta el *cutoff*, la variante personalizada mejora sus evaluaciones de precisión llegando en el cutoff 50 a situarse como el primer algoritmo en lo que a precisión respecta y como el segundo mejor algoritmo si lo valoramos con *nDCG*. Esta tendencia del algoritmo *PageRank Personalizado* ya la habíamos observado en el conjunto de datos *follow*, no obstante, en el conjunto *mention / reply / retweet* se manifiesta de forma más contundente.

El algoritmo *MaxFOAF*, que en el conjunto de datos anterior se perfilaba como el más efectivo de los algoritmos individuales se ve relegado a posiciones peores en el escenario presente. Esto puede ser debido a que la menor densidad del grafo *Mention / Reply /Retweet* propicie en este algoritmo el problema conocido comúnmente como *cold-start* o arranque frío.

En lo que respecta al algoritmo Popularity su posicionamiento en lo que a precisión respecta dentro del conjunto de algoritmos de recomendación es similar al que se ha observado en el conjunto de datos Follow. Lo mismo sucede con el algoritmo *Betweenness* y el algoritmo *HITS Hub*. En particular este último se mantiene al fondo del ranking de precisión llegando a obtener evaluaciones peores aún que las recibidas por el algoritmo línea base *Random*.

En lo que respecta al algoritmo basado en contenido Rocchio en sus dos variantes, los resultados de *precisión* y *nDCG* son realmente discretos. Llegan a superar a los algoritmos *HITS Hub* y *Random* confirmando que realizan una recomendación no trivial pero no llegan a alcanzar los resultados de los algoritmos basados en red social. No obstante no deberíamos caer en el engaño de considerar a tenor de estos resultados que los algoritmos basados en contenido son menos útiles que los basados en red social: dos usuarios que hablan de temas similares no necesariamente tiene que haberse conocido necesariamente antes de la construcción del conjunto de datos. La forma más apropiada de evaluar la utilidad real de estos algoritmos pararía por tanto por recomendar a un usuario real usuarios haciendo uso de los algoritmos basados en contenido y preguntar al usuario por la utilidad de las recomendaciones. De todas formas, volviendo a centrarnos en las métricas estudiadas, sí que presenta evaluaciones ligeramente superiores la variante que penaliza los *tweets* más antiguos durante la creación de los *centroides* representativos de los usuarios. Esto último puede estar relacionado con el hecho de que parece más probable que se conozcan dos usuarios que hablan de lo mismo si lo hacen al mismo tiempo.

Queda por último comentar los resultados obtenidos por los algoritmos híbridos que combinan los algoritmos *MaxFOAF*, *Popularity* o *PageRank* con las dos variantes del

algoritmo *Rocchio*. Como podemos observar en la tabla 15, sólo la combinación con *MaxFOAF* consigue mejorar la precisión en *cutoff* 5 de los algoritmos involucrados en la hibridación. En este caso, al igual que sucedía de manera individual con las dos variantes de *Rocchio*, las evaluaciones de la versión en la que se penalizan los tweets más antiguos obtiene evaluaciones ligeramente mejores.

6.2.2.2 Observaciones acerca de los resultados de novedad y diversidad

De nuevo el algoritmo *Random* se presenta como el algoritmo más destacado en términos de novedad por los mismos motivos que en el conjunto de datos *follow* y a pesar de ser uno de los que más alta diversidad presenta llega a verse superado por las dos variantes del algoritmo *Rocchio*. Esto último refuerza la hipótesis planteada cuando comentábamos la baja precisión que presentaba este algoritmo.

El algoritmo *Popularity* de nuevo es el que minimiza la métrica de novedad, la explicación de este hecho es la misma que se ha ofrecido en el conjunto de datos *follow*. También como sucedía antes, el algoritmo MaxFOAF es uno de los que más novedad consigue registrar.

Una observación más detenida nos lleva a darnos cuenta de que el posicionamiento de los algoritmos basados en la red social observados previamente en el conjunto de datos *follow* se mantiene en la línea que se mantenía en el conjunto de datos anterior. No obstante, en este caso particular, al haber variado la ordenación de los algoritmos por precisión, sí que observamos en la figura 36 una cierta proporcionalidad inversa entre precisión y novedad (salvo excepciones como el algoritmo *MaxFOAF* tanto en solitario como en combinación con *Rocchio*) que no observábamos de forma tan clara en la figura 35.

En lo que respecta a diversidad, no llegamos a observar una proporcionalidad tan clara, no obstante, al igual que sucedía con novedad, que el posicionamiento de los algoritmos basados en la red social observados previamente en el conjunto de datos *follow* se mantiene en la línea que se mantenía en el conjunto de datos anterior.

7. Conclusiones

Para concluir este trabajo de fin de grado, vamos a recapitular analizando la consecución de los objetivos propuestos en el apartado 1.2 de la introducción así como las contribuciones más interesantes del trabajo realizado. Por último, vamos a presentar distintas líneas de trabajo futuro que pueden partir de la base de las herramientas desarrolladas y los experimentos realizados en el contexto de este trabajo de fin de grado.

7.1 Resumen y contribuciones

En este trabajo de fin de grado se ha diseñado e implementado una plataforma de experimentación completa y flexible que ha permitido la preparación de un conjunto de datos de *Twitter* así como la experimentación con algoritmos de recomendación de usuarios. De esta manera quedan cubiertos los dos primeros objetivos planteados al inicio y se han generado herramientas de experimentación idóneas para realizar nuevos experimentos de recomendación basados en *Twitter* tal y como se expondrá en el próximo sub-apartado.

En el marco de la plataforma desarrollada y tomando como base el conjunto de datos preparado, se han implementado algoritmos de recomendación de usuarios basados por una parte en la topología de red social, por otra parte en el contenido de los tweets publicados por los usuarios llegando a tomar en consideración la variable temporal en este enfoque y por último se ha desarrollado un sistema de hibridación de algoritmos que ha permitido combinar distintos algoritmos para generar nuevos algoritmos híbridos. De este modo queda cubierto el tercero de los objetivos planteados.

La plataforma de experimentación desarrollada nos ha permitido también evaluar los algoritmos de recomendación de usuarios definidos e implementados desde dos enfoques distintos: la red social explícita representativa de las interacciones *follow* entre usuarios y la red social implícita representativa de las interacciones *mention*, *reply* y *retweet*. En todos nuestros experimentos hemos utilizado métricas de precisión, novedad y diversidad que han permitido comparar desde distintos puntos de vista los diferentes algoritmos implementados. De este modo queda cubierto el cuarto y último de los objetivos planteados.

Dentro del conjunto de algoritmos con los que hemos podido experimentar desde los dos enfoques de red social (es decir, aquellos que utilizan como input únicamente en el grafo dirigido representativo de la red social) hemos podido observar que la efectividad de los mismos varía en función del tipo de interacciones tomadas en consideración para modelar la red social. Por ejemplo, a la hora de evaluar la precisión en la recomendación de listas de hasta 5 ítems para cada usuario, el algoritmo *HITS Authority* resulta ser el que mejores resultados ha presentado en el grafo *mention / reply / retweet* mientras que en el grafo *follow* ha sido el híbrido resultante de la combinación de los algoritmos *MaxFOAF* con peso 0.6 y *Popularity* con peso 0.4 el que mejor evaluación ha obtenido en este tipo de experimentos.

También hemos podido observar empíricamente que en ciertos escenarios la hibridación de algoritmos puede dar lugar a algoritmos más efectivos. Destaca en este sentido la combinación de los algoritmos *MaxFOAF* y *Popularity* en el grafo *follow*, que consigue los mejores resultados de precisión en todos los experimentos realizados en dicho grafo así como en buena parte de los experimentos realizados en el grafo *mention* / *reply* / *retweet*.

El conjunto de datos preparado ha sido también de utilidad dentro del ámbito del laboratorio del Grupo de Recuperación de Información ya que a día de hoy ha sido empleado como fuente de datos de ejemplo en otros dos trabajos de fin de grado en los que se han podido analizar distintas características de la red social así como abordar la recomendación de *tweets* en lugar de usuarios. Además ha sido de utilidad para proveer un ejemplo de grafo dirigido de red social en la última práctica de la asignatura de búsqueda y minería de información en 4º del Grado en Ingeniería Informática en la UAM.

La plataforma de experimentación en conjunto con el conjunto de datos preparado ha permitido también aproximarse a la experimentación de algoritmos de recomendación avaros orientados a la optimización de distintas métricas de novedad y diversidad tratando de minimizar la pérdida de precisión. Esta pequeña investigación se ha llevado a cabo en paralelo al desarrollo de este trabajo de fin de grado concluyendo en la publicación del póster (Alhambra, et al., 2014) en el III Congreso Español de Recuperación de Información (CERI 2014) que se celebrará en A Coruña en Junio de este año.

7.2 Trabajo futuro

Como hemos podido observar a lo largo de todo el trabajo de fin de grado, se ha trabajado con la motivación de que la labor realizada en este contexto no se limite como fin en sí mismo sino que deje abiertas distintas posibilidades de investigación.

Las líneas más inmediatas de trabajo futuro son aquellas orientadas a ampliar el abanico de algoritmos de recomendación de usuarios con los que experimentar así como incrementar la variedad de métricas de evaluación de algoritmos. Esta tarea encaja de forma idónea con la plataforma de experimentación desarrollada dado que su realización se limitaría a la implementación de interfaces de recomendación y evaluación que los distintos programas de la plataforma de experimentación puedan utilizar modificando simplemente sus parámetros de ejecución. En este contexto, una de las líneas más inmediatas podría ser la combinación específica de distintos algoritmos estudiados. Un ejemplo de este tipo de combinación específica podría ser el de combinar *MaxFOAF* y *HITS Hub* utilizando los scores asignados por el segundo a los usuarios seguidos por un usuario para ponderar los scores que asigna *MaxFOAF* a los usuarios seguidos por estos usuarios.

Otra línea relativamente inmediata de trabajo futuro es la de abordar la recomendación de diferentes ítems tales como *tweets*, URLs o incluso *hashtags*. Dada la flexibilidad de la plataforma de recomendación, para hacer esto sólo sería necesario implementar distintos algoritmos en la interfaz de separación de datos que permitan crear juicios de relevancia en los que los identificadores de los ítems relevantes ya no sean de usuarios sino

de los distintos tipos de ítems susceptibles de ser recomendados. Hecho esto, al igual que en la línea de trabajo futuro anterior, bastaría la implementación de las interfaces de recomendación y evaluación para poder realizar nuevos experimentos.

Una línea de trabajo futuro más ambiciosa y compleja consiste en la ampliación del conjunto de datos preparado recuperando información del repositorio JSON para ampliar la riqueza de la estructura de información a la que tienen acceso los algoritmos de recomendación permitiendo así la experimentación con algoritmos de recomendación capaces de considerar más variables en su tarea.

No podemos cerrar el apartado de trabajo futuro sin mencionar la línea de trabajo más compleja pero realista que es la que pone a prueba realmente los sistemas de recomendación con los que se ha experimentado: la experimentación en tiempo real con usuarios reales para evaluar los distintos sistemas de recomendación con las métricas más fiables posibles a la hora de valorar la utilidad de un sistema de recomendación que son las encuestas de satisfacción de los propios usuarios finales del mismo.

Bibliografía

Adomavicius, G. & Tuzhilin, A. Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions. *IEEE Transactions on Knowledge and Data Engineering,* 17(6), Junio 2005, pp. 734-749.

Alhambra, A., Pepa, S. M. & Castells, P. Recomendación en redes sociales: novedad y diversidad. *III Congreso Español de Recuperación de Información (CERI 2014),* Junio 2014.

Baeza, R. & Ribeiro, B. *Mordern Informations Retrieval.* Addison-Wesley, 1999.

D. Blondel, V., Guillaume, J.-L., Lambiotte, R. & Lefebvre, E. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment* 10, 2008.

Fernández, P.. El secreto de Google y el Álgebra lineal. En: *Boletín de la Sociedad Española de Matemática Aplicada* 30, 2004, pp. 115-141.

Fix, E. & Hodges, J. An Important Contribution to Nonparametric Discriminant Analysis and Density Estimation: Commentary on Fix and Hodges. *International Statistical Review / Revue Internationale de Statistique,* Vol. 57, No. 3 Diciembre 1951, pp. 233-238.

Jarvelin, K. & Kekalainen, J. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems* 20(4), 2001, p. 422–446.

Kleinberg, J. Authoritative sources in a hyperlinked environment. *Journal of the ACM* 46, Septiembre 1999, pp. 604–632.

Koren, Y., Bell, R. & Volinsky, C. Matrix Factorization Techniques for Recommender Systems. *IEEE Computer Society,* 42(8), Agosto 2009, pp. 30-37.

Kywe, S. M., Lim, E.-P. & Zhu, F., 2012. A Survey of Recommender Systems in Twitter. *SocInfo, Lecture Notes in Computer Science,* Vol, 7710 pp 420-433. Springer, 2012.

Page, L. & Brin, S., 1998. *The anatomy of a large-scale hypertextual Web search engine.* Brisbane, Australia, s.n.

Ricci, F., Rokach, L., Shapira, B. & Kantor, P. B. *Recommender Systems Handbook.* Springer, 2011.

Rocchio, J. Relevance Feedback in Information Retrieval. En: G. Salton, ed. *The SMART System: Experiments in Automatic Document Processing.* Prentice Hall, 1971, pp. 313-323.

Russell, M. A. *Mining the SocialWeb.* O'Reilly, 2011.

Wasserman, S. & Faust, K.. Social Network Analysis: Methods and Applications. En: *Social Network Analysis in the Social and Behavioral Sciences.* Cambridge University Press, 1994, pp. 1-27.

White, S. & Smyth, P., 2003. *Algorithms for estimating relative importance in networks. 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2003)*. Washington, D.C., USA, August 2003, pp. 266-275

1. Anexo I: Código SQL del conjunto de datos escenario

En el presente anexo se presenta el código SQL de la base de datos que modela el conjunto de datos empleado para recomendación en Twitter cuyo diagrama ER se ha presentado en las figuras 13 y 14. Se presenta por una parte el SQL de la base de datos lista para inserción de datos (prescindiendo de parte de las restricciones de integridad por clave foránea) y a continuación se presenta el código de las claves foráneas que se insertan una vez aislado el conjunto de datos garantizando así la integridad referencial del conjunto de datos final.

1.1 Base de datos preparada para inserción de datos

El código SQL que se presenta a continuación es el que genera la base de datos del conjunto de datos escenario preparada para la inserción de datos por parte de múltiples instancias del programa de generación del conjunto de datos. Los únicos datos que se insertan directamente en este código son los valores de las tablas que modelan campos enumerados.

```
-- Settings table creation
CREATE TABLE IF NOT EXISTS `Settings` (
  `acceptance_min_tweets_per_user` bigint(20) unsigned NOT NULL COMMENT 'Mini-
mum number of tweets a user should have publishd to be accepted in the da-
taset.',
  `acceptance_min_followees_per_user` bigint(20) unsigned NOT NULL COMMENT
'Minimum number of followees a user should have to be accepted in the da-
taset.',
  `acceptance_max_followees_per_user` bigint(20) unsigned NOT NULL COMMENT
'Maximum number of followees a user should have to be accepted in the da-
taset.',
  `acceptance_min_time_since_user_creation` bigint(20) unsigned NOT NULL COM-
MENT 'Minimum number of  milliseconds betweet the user creation and the user
insertion in the dataset for a user to be accepted in the dataset.',
  `expansion_number_of_users_to_collect` bigint(20) unsigned NOT NULL COMMENT
'Number of users to collect in the dataset.',
  `expansion_max_number_of_tweets_to_retrieve_per_user` int(10) unsigned NOT
NULL COMMENT 'Maximum number of tweets to collect per user.',
  `expansion_max_number_of_sons_to_expand_per_user` bigint(20) unsigned NOT
NULL COMMENT 'Maximum number of followees to collect per user.',
  `expansion_root_user_id` bigint(20) unsigned NOT NULL COMMENT 'Root user
Id.'
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin;

-- SocialLinkDated table creation
CREATE TABLE IF NOT EXISTS `SocialLinkDated` (
  `source_user_id` bigint(20) unsigned NOT NULL COMMENT 'Directed graph''s
link''s source user''s Id.',
```

```
  `destination_user_id` bigint(20) unsigned NOT NULL COMMENT 'Directed
graph''s link''s destination user''s Id.',
  `type_id` tinyint(4) unsigned NOT NULL COMMENT 'Link type: 0 FOLLOW, 1 MEN-
TION, 2 REPLY, 3 RETWEET.',
  `date` bigint(20) unsigned NOT NULL COMMENT 'Link creation date.',
  PRIMARY KEY (`source_user_id`,`destination_user_id`,`type_id`,`date`),
  KEY `source_user_id` (`source_user_id`),
  KEY `destination_user_id` (`destination_user_id`),
  KEY `type_id` (`type_id`),
  KEY `date` (`date`),
  CONSTRAINT `FK_SocialLinkDated_SocialLinkType` FOREIGN KEY (`type_id`) REF-
ERENCES `SocialLinkType` (`type_id`) ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `FK_SocialLinkDated_User` FOREIGN KEY (`source_user_id`) REFER-
ENCES `User` (`user_id`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin;

-- SocialLinkType table creation
DROP TABLE IF EXISTS `SocialLinkType`;
CREATE TABLE IF NOT EXISTS `SocialLinkType` (
  `type_id` tinyint(4) unsigned NOT NULL COMMENT 'Link type Id.',
  `type` varchar(15) COLLATE utf8_bin NOT NULL COMMENT 'Link type.',
  PRIMARY KEY (`type_id`),
  KEY `type` (`type`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin;

-- SocialLinkType table data: 4 rows
INSERT INTO `SocialLinkType` (`type_id`, `type`) VALUES
       (0, 'FOLLOW'),
       (1, 'MENTION'),
       (2, 'REPLY'),
       (3, 'RETWEET');

-- SocialLinkWeighted table creation
CREATE TABLE IF NOT EXISTS `SocialLinkWeighted` (
  `source_user_id` bigint(20) unsigned NOT NULL COMMENT 'Directed graph''s
link''s source user''s Id.',
  `destination_user_id` bigint(20) unsigned NOT NULL COMMENT 'Directed
graph''s link''s destination user''s Id.',
  `type_id` tinyint(4) unsigned NOT NULL COMMENT 'Link type: 0 FOLLOW, 1 MEN-
TION, 2 REPLY, 3 RETWEET.',
  `weight` int(10) unsigned NOT NULL DEFAULT '1' COMMENT 'Link weight, for
FOLLOW links the only possible value is 1, for the rest this number represents
the number of interactions produced.',
  PRIMARY KEY (`source_user_id`,`destination_user_id`,`type_id`),
  KEY `source_user_id` (`source_user_id`),
  KEY `destination_user_id` (`destination_user_id`),
  KEY `type_id` (`type_id`),
  CONSTRAINT `FK_SocialLinkWeighted_SocialLinkType` FOREIGN KEY (`type_id`)
REFERENCES `SocialLinkType` (`type_id`) ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `FK_SocialLinkWeighted_User` FOREIGN KEY (`source_user_id`) REF-
ERENCES `User` (`user_id`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin;

-- Tweet table creation
CREATE TABLE IF NOT EXISTS `Tweet` (
  `tweet_id` bigint(20) unsigned NOT NULL COMMENT 'Tweet''s Id.',
  `text` varchar(200) COLLATE utf8_bin NOT NULL COMMENT 'Tweet''s text.',
  `date` bigint(20) NOT NULL COMMENT 'Tweet''s creation date.',
  `user_id` bigint(20) unsigned NOT NULL COMMENT 'Tweet''s author''s Id.',
```

```
  `favorite_count` bigint(20) unsigned NOT NULL COMMENT 'Number of users who
selected this tweet as favourite.',
  `retweet_count` bigint(20) unsigned NOT NULL COMMENT 'Number of times this
tweet has been retweeted.',
  PRIMARY KEY (`tweet_id`),
  KEY `user_id` (`user_id`),
  KEY `favourite_count` (`favorite_count`),
  KEY `retweet_count` (`retweet_count`),
  CONSTRAINT `FK_Tweet_User` FOREIGN KEY (`user_id`) REFERENCES `User`
(`user_id`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin;

-- TweetActionMention table creation
CREATE TABLE IF NOT EXISTS `TweetActionMention` (
  `tweet_id` bigint(20) unsigned NOT NULL COMMENT 'Id identifying the tweet in
which the user is mentioned.',
  `mentioned_user_id` bigint(20) unsigned NOT NULL COMMENT 'Mentioned user''s
Id.',
  PRIMARY KEY (`tweet_id`,`mentioned_user_id`),
  KEY `tweet_id` (`tweet_id`),
  KEY `mentioned_user_id` (`mentioned_user_id`),
  CONSTRAINT `FK_TweetActionMention_Tweet` FOREIGN KEY (`tweet_id`) REFERENCES
`Tweet` (`tweet_id`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin;

-- TweetActionReply table creation
CREATE TABLE IF NOT EXISTS `TweetActionReply` (
  `answer_tweet_id` bigint(20) unsigned NOT NULL COMMENT 'Answering tweet''s
Id.',
  `question_tweet_id` bigint(20) unsigned NOT NULL COMMENT 'Replied tweet''s
Id.',
  `question_user_id` bigint(20) unsigned NOT NULL COMMENT 'Replied tweet''s
author Id.',
  PRIMARY KEY (`answer_tweet_id`),
  KEY `question_tweet_id` (`question_tweet_id`),
  KEY `answer_tweet_id` (`answer_tweet_id`),
  KEY `question_user_id` (`question_user_id`),
  CONSTRAINT `FK_TweetActionReply_Tweet` FOREIGN KEY (`answer_tweet_id`) REF-
ERENCES `Tweet` (`tweet_id`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin;

--.TweetActionRetweet table creation
CREATE TABLE IF NOT EXISTS `TweetActionRetweet` (
  `retweeter_tweet_id` bigint(20) unsigned NOT NULL COMMENT 'Retweet tweet
Id.',
  `retweeted_tweet_id` bigint(20) unsigned NOT NULL COMMENT 'Retweeted tweet
Id.',
  `retweeted_user_id` bigint(20) unsigned NOT NULL COMMENT 'Retweeted tweet''s
author''s Id.',
  PRIMARY KEY (`retweeter_tweet_id`),
  KEY `retweeter_tweet_id` (`retweeter_tweet_id`),
  KEY `retweeted_tweet_id` (`retweeted_tweet_id`),
  KEY `retweeted_user_id` (`retweeted_user_id`),
  CONSTRAINT `FK_TweetActionRetweet_Tweet` FOREIGN KEY (`retweeter_tweet_id`)
REFERENCES `Tweet` (`tweet_id`) ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `FK_TweetActionRetweet_Tweet_2` FOREIGN KEY (`re-
tweeted_tweet_id`) REFERENCES `Tweet` (`tweet_id`) ON DELETE CASCADE ON UPDATE
CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin;
```

```
-- TweetComponentHashtag table creation
CREATE TABLE IF NOT EXISTS `TweetComponentHashtag` (
  `tweet_id` bigint(10) unsigned NOT NULL COMMENT 'Id identifying the tweet in
which the hashtag appeared.',
  `hashtag` varchar(50) COLLATE utf8_bin NOT NULL COMMENT 'Hashtag text.',
  PRIMARY KEY (`tweet_id`,`hashtag`),
  KEY `hashtag` (`hashtag`),
  KEY `tweet_id` (`tweet_id`),
  CONSTRAINT `FK_TweetComponentHashtag_Tweet` FOREIGN KEY (`tweet_id`) REFER-
ENCES `Tweet` (`tweet_id`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin;

-- TweetComponentUrl table creation
CREATE TABLE IF NOT EXISTS `TweetComponentUrl` (
  `tweet_id` bigint(20) unsigned NOT NULL COMMENT 'Id identifying the tweet in
which the url appeared.',
  `url` varchar(128) COLLATE utf8_bin NOT NULL COMMENT 'Original url.',
  `expanded_url` varchar(512) COLLATE utf8_bin NOT NULL COMMENT 'Expanded
url.',
  `display_url` varchar(128) COLLATE utf8_bin NOT NULL COMMENT 'Displayed
url.',
  PRIMARY KEY (`tweet_id`,`url`),
  KEY `url` (`url`),
  KEY `expanded_url` (`expanded_url`(255)),
  KEY `display_url` (`display_url`),
  CONSTRAINT `FK_TweetComponentUrl_Tweet` FOREIGN KEY (`tweet_id`) REFERENCES
`Tweet` (`tweet_id`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin;

-- User table creation
CREATE TABLE IF NOT EXISTS `User` (
  `user_id` bigint(20) unsigned NOT NULL COMMENT 'User Id.',
  `screen_name` varchar(30) COLLATE utf8_bin NOT NULL COMMENT 'User''s Screen
Name (that oune usually preceded by arroba).',
  `name` varchar(30) COLLATE utf8_bin NOT NULL COMMENT 'Complet user name.',
  `date` bigint(20) unsigned NOT NULL COMMENT 'Twitter user account creation
date.',
  `verified` tinyint(4) unsigned NOT NULL COMMENT 'User verified as a celeb-
rity.',
  `num_tweets` bigint(20) unsigned NOT NULL COMMENT 'Number of tweets the user
has published.',
  `num_followers` bigint(20) unsigned NOT NULL COMMENT 'Number of users who
follow the user.',
  `num_followees` bigint(20) unsigned NOT NULL COMMENT 'Number of users the
user follows.',
  `generation` tinyint(4) unsigned NOT NULL COMMENT 'User''s generation in the
social net graph.',
  `status_id` tinyint(4) unsigned NOT NULL COMMENT 'Tweets collection status
flag.',
  `expanded_id` tinyint(4) unsigned NOT NULL COMMENT 'Expansion status flag.',
  `parent_user_id` bigint(20) unsigned NOT NULL COMMENT 'Social net graph par-
ent user Id (The user who was expanded to insert this user in the dataset). 0
In case of the root user or a collected user.',
  `insertion_date` bigint(20) unsigned NOT NULL COMMENT 'Date when the user
was inserted in the dataset.',
  PRIMARY KEY (`user_id`),
  KEY `name` (`name`) USING HASH,
```

```
  KEY `screen_name` (`screen_name`),
  KEY `insertion_date` (`insertion_date`),
  KEY `FK_User_UserStatus` (`status_id`),
  KEY `FK_User_UserExpanded` (`expanded_id`),
  KEY `FK_User_User` (`parent_user_id`),
  CONSTRAINT `FK_User_UserExpanded` FOREIGN KEY (`expanded_id`) REFERENCES
`UserExpanded` (`expanded_id`) ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `FK_User_UserStatus` FOREIGN KEY (`status_id`) REFERENCES `Us-
erStatus` (`status_id`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin;

-- UserExpanded table creation
DROP TABLE IF EXISTS `UserExpanded`;
CREATE TABLE IF NOT EXISTS `UserExpanded` (
  `expanded_id` tinyint(4) unsigned NOT NULL COMMENT 'Expansion information
Id.',
  `expanded` varchar(20) COLLATE utf8_bin NOT NULL COMMENT 'Expansion infor-
mation.',
  PRIMARY KEY (`expanded_id`),
  KEY `expanded_id` (`expanded_id`),
  KEY `expanded` (`expanded`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin;

-- UserExpanded table data: 10 rows
INSERT INTO `UserExpanded` (`expanded_id`, `expanded`) VALUES
       (5, 'BACKBONE_EXPANDED'),
       (4, 'BACKBONE_EXPANDING'),
       (3, 'BACKBONE_TO_EXPAND'),
       (8, 'LEAF_EXPANDED'),
       (7, 'LEAF_EXPANDING'),
       (6, 'LEAF_TO_EXPAND'),
       (9, 'NOT_EXPANDABLE'),
       (2, 'ROOT_EXPANDED'),
       (1, 'ROOT_EXPANDING'),
       (0, 'ROOT_TO_EXPAND');

-- UserStatus table creation
CREATE TABLE IF NOT EXISTS `UserStatus` (
  `status_id` tinyint(4) unsigned NOT NULL COMMENT 'Status id.',
  `status` varchar(20) COLLATE utf8_bin NOT NULL COMMENT 'Status.',
  PRIMARY KEY (`status_id`),
  KEY `status_id` (`status_id`),
  KEY `status` (`status`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin;

-- UserStatus table data: 5 rows
DELETE FROM `UserStatus`;
INSERT INTO `UserStatus` (`status_id`, `status`) VALUES
       (0, 'ACCEPTED'),
       (1, 'COLLECTED'),
       (2, 'DISCARDED'),
       (3, 'PROCESING'),
       (4, 'PROCESSED');
```

1.2 Integridad referencial en la base de datos final

Una vez sea ejecutado el programa de aislamiento de la base de datos, la integridad referencial total del conjunto de datos aislado final queda garantizada añadiendo las restricciones de clave foránea que aparecen en el siguiente código SQL que se presenta en esta sub-sección. Estas restricciones son incompatibles con el procedimiento de inserción de los datos en el conjunto de datos pero sí son compatibles con el estado de la base de datos final una vez se ha ejecutado el programa *DbIsolator*. Estas restricciones son útiles especialmente a la hora de realizar consultas ya que generan índices que optimizan las mismas.

```
-- SocialLinkDated constraints
ALTER IGNORE TABLE `SocialLinkDated` ADD CONSTRAINT `FK_Social-
LinkDated_User_2` FOREIGN KEY (`destination_user_id`) REFERENCES `User`
(`user_id`) ON UPDATE CASCADE ON DELETE CASCADE;

-- SocialLinkWeighted constraints
ALTER IGNORE TABLE `SocialLinkWeighted` ADD CONSTRAINT `FK_Social-
LinkWeighted_User_2` FOREIGN KEY (`destination_user_id`) REFERENCES `User`
(`user_id`) ON UPDATE CASCADE ON DELETE CASCADE;

-- TweetActionMention constraints
ALTER IGNORE TABLE `TweetActionMention` ADD CONSTRAINT `FK_TweetActionMen-
tion_User` FOREIGN KEY (`mentioned_user_id`) REFERENCES `User` (`user_id`) ON
UPDATE CASCADE ON DELETE CASCADE;

-- TweetActionReply constraints
ALTER IGNORE TABLE `TweetActionReply` ADD CONSTRAINT `FK_TweetActionRe-
ply_Tweet_2` FOREIGN KEY (`question_tweet_id`) REFERENCES `Tweet` (`tweet_id`)
ON UPDATE CASCADE ON DELETE CASCADE;

ALTER IGNORE TABLE `TweetActionReply` ADD CONSTRAINT `FK_TweetActionRe-
ply_User` FOREIGN KEY (`question_user_id`) REFERENCES `User` (`user_id`) ON
UPDATE CASCADE ON DELETE CASCADE;

-- TweetActionRetweet constraints
ALTER IGNORE TABLE `TweetActionRetweet` ADD CONSTRAINT `FK_TweetAction-
Retweet_User` FOREIGN KEY (`retweeted_user_id`) REFERENCES `User` (`user_id`)
ON UPDATE CASCADE ON DELETE CASCADE;

-- User constraints
ALTER IGNORE TABLE `User` ADD CONSTRAINT `FK_User_User` FOREIGN KEY (`par-
ent_user_id`) REFERENCES `User` (`user_id`) ON UPDATE CASCADE ON DELETE CAS-
CADE;
```

2. Anexo II: Descripción detallada conjunto de datos escenario

En el presente anexo se especifica la estructura de las tablas del conjunto de datos así como las relaciones entre las mismas. La información presente en este anexo complementa la información presentada en el anexo 1 y las figuras 13 y 14.

2.1 Tablas del conjunto de datos

2.1.1 User

La tabla *User* modela los usuarios de *Twitter* presentes en el conjunto de datos, para ello cuenta con los siguientes campos:

- **user_id**: Identificador numérico unívoco de los usuarios generado y provisto por *Twitter*. Este campo es la clave primaria de la tabla *User* y se almacena como un entero de 8 bytes.
- **screen_name**: Identificador textual unívoco de los usuarios elegido por los mismos en la creación de su cuenta de *Twitter*. Este campo coincide con el nombre de usuario que se presenta precedido por el caracter '@' en la interfaz web de *Twitter* y es el empleado por los usuarios en las búsquedas y menciones de otros usuarios. A diferencia del anterior identificador que permanece inalterable a lo largo de la vida de una cuenta de *Twitter*, este campo puede ser modificado por el usuario en cualquier momento. En la base de datos se almacena el valor de este campo en el momento de la recolección del usuario como campo de texto de longitud variable en codificación utf8_bin.
- **name**: Identificador textual extendido de los usuarios elegidos por los mismos en la creación de su cuenta de *Twitter*. Este campo coincide con el nombre de usuario presentado en la página de perfil de un usuario de *Twitter* y, al igual que sucede con el campo screen_name, este valor puede ser modificado por el usuario en cualquier momento. En la base de datos se almacena el valor de este campo en el momento de la recolección del usuario como campo de texto de longitud variable en codificación utf8_bin.
- **date**: Fecha de creación del perfil de usuario en *Twitter*. Este campo se genera en el momento en el que el usuario accede a *Twitter* por primera vez y crea su cuenta y se almacena en la tabla **User** como un entero de 8 bytes representando los milisegundos transcurridos desde 1970-01-01 00:00:00 GMT. Esta fecha se emplea durante la creación del conjunto de datos para determinar si un usuario tiene una antigüedad mínima para ser considerador como un componente del conjunto de datos.

- **verified**: Indicador que permite diferenciar en *Twitter* cuentas oficiales y verificadas de cuentas particulares. En la interfaz web de *Twitter* esta información se manifiesta en forma de un sello azul con un tick blanco asociado a los usuarios verificados. Este indicador se almacena en la tabla user como un entero de un byte representando el valor 0 un usuario no verificado y el valor 1 un usuario verificado.
- **num_tweets**: Contador que indica el número de tweets publicados por el usuario en la red social *Twitter* en el momento de su inserción en la base de datos. Este contador se almacena en la tabla como un entero no negativo y se emplea durante la creación del conjunto de datos para determinar si un usuario satisface un umbral mínimo de actividad en *Twitter* para ser considerado como un componente del conjunto de datos.
- **num_followers**: Contador que indica el número de usuarios de *Twitter* que siguen al usuario en el momento de su inserción en la base de datos. Este contador se almacena en la tabla como un entero no negativo y se emplea durante la creación del conjunto de datos para determinar si un usuario reúne un número mínimo de seguidores para ser considerado como un componente del conjunto de datos.
- **num_followees**: Contador que indica el número de usuarios de *Twitter* que el usuario sigue en el momento de su inserción en la base de datos. Este contador en la tabla como un entero no negativo y se emplea durante la creación del conjunto de datos para comprobar que un usuario no excede un número máximo de usuarios seguidos para ser considerado como un componente del conjunto de datos.
- **generation**: Contador que indica la longitud del camino más corto avanzando por usuarios raíz entre el primer usuario insertado en el conjunto de datos (el usuario raíz del conjunto de datos) y el usuario.
- **status**: Marcador enumerado del estado del usuario en la base de datos. Se almacena como una referencia de clave foránea a la tabla de estados aceptados *UserStatus*. Los estados posibles son:
    0) **ACCEPTED**: El usuario ha sido aceptado como componente del conjunto de datos y es apto para que sus tweets sean añadidos al conjunto de datos.
    1) **COLLECTED**: El usuario ha sido insertado en la base de datos debido a que ha aparecido como autor de un *tweet retwitteado* por un usuario del conjunto de datos. El usuario aún no ha pasado los filtros de aceptación y consecuentemente no está preparado aún para que sus tweets sean añadidos al conjunto de datos.
    2) **DISCARDED**: El usuario ha sido insertado en la base de datos como usuario seguido de un usuario backbone (o bien había sido insertado previamente como autor de un tweet retwitteado por un usuario del conjunto de datos y posteriormente evaluado como usuario seguido por un usuario backbone) pero no ha cumplido los requisitos para ser aceptado como componente del conjunto de datos (antigüedad inferior a la mínima aceptada, numero de tweets publicados inferior al mínimo aceptado, número de seguidores inferior al mínimo aceptado o número de usuarios seguidos superior al máximo aceptado).

3) **PROCESSING**: Usuario aceptado (su anterior valor en este marcador era ACCEPTED) cuyos tweets están siendo recolectados por el programa de generación del conjunto de datos. Este valor es necesario para la gestión de la concurrencia entre distintas máquinas ejecutando el programa de generación del conjunto de datos. Las razones de la necesidad de esta concurrencia quedan recogidas en el siguiente apartado de generación del conjunto de datos.
4) **PROCESSED**: Usuario aceptado final del conjunto de datos. Todos sus tweets (hasta el máximo determinado en el programa de generación del conjunto de datos) han sido insertados en el conjunto de datos.

- **expanded**: Marcador enumerado del estado de expansión de un usuario en la base de datos. Se almacena como una referencia de clave foránea a la tabla de estados de expansión aceptados *UserExpanded*. Se define expandir un usuario como recolectar la información de usuario de todos los usuarios que sigue, evaluar los campos de aceptación de los mismos y asignarles el status ACCEPTED o DISCARDED y almacenarlos en la base de datos. Los estados de expansión posibles son:
  0) **ROOT_TO_EXPAND**: *Usuario raíz* del conjunto de datos no expandido y apto para su expansión. El *usuario raíz* es el primer usuario insertado en el conjunto de datos. Éste admite un único usuario raíz.
  1) **ROOT_EXPANDING**: *Usuario raíz* en proceso de expansión. Este valor es necesario para la gestión de la concurrencia entre distintas máquinas ejecutando el programa de generación del conjunto de datos.
  2) **ROOT_EXPANDED**: *Usuario raíz* expandido: Ha sido descargada de *Twitter* la información de usuario de todos sus followees y éstos han sido insertados en el conjunto de datos o descartados.
  3) **BACKBONE_TO_EXPAND**: *Usuario rama* del conjunto de datos no expandido y apto para su expansión. Los *usuarios rama* son aquellos empleados por el programa de generación del conjunto de datos para incluir más usuarios en el conjunto de datos.
  4) **BACKBONE_EXPANDING**: *Usuario rama* en proceso de expansión. Este valor es necesario para la gestión de la concurrencia entre distintas máquinas ejecutando el programa de generación del conjunto de datos.
  5) **BACKBONE_EXPANDED**: *Usuario rama* expandido: Ha sido descargada de *Twitter* la información de usuario de todos sus followees y éstos han sido insertados en el conjunto de datos o descartados.
  6) **LEAF_TO_EXPAND**: Usuario hoja del conjunto de datos no expandido y apto para su expansión. Los *usuarios hoja* son aquellos que no se emplean por el programa de generación del conjunto de datos para incluir más usuarios en el conjunto de datos. No obstante, es necesario descargar de *Twitter* la información de usuario de todos sus followees para insertar en el conjunto de datos final los links sociales de los usuarios hoja con otros usuarios del conjunto de datos.
  7) **LEAF_EXPANDING**: *Usuario hoja* en proceso de expansión. Este valor es necesario para la gestión de la concurrencia entre distintas máquinas ejecutando el programa de generación del conjunto de datos.

8) **LEAF_EXPANDED**: *Usuario hoja* expandido: Ha sido descargada de *Twitter* la información de usuario de todos sus followees y en el conjunto de datos se ha almacenado a quien sigue el usuario hoja.
9) **NOT_EXPANDABLE**: Usuario no expandible. Este valor es el que se asigna a los usuarios con valores de **status** COLLECTED o DISCARDED.

2.1.2 Tweet

La tabla *Tweet* modela los *tweets* de *Twitter* publicados por los usuarios y presentes en el conjunto de datos, para ello cuenta con los siguientes campos:

- **tweet_id**: Identificador numérico unívoco de los tweets generado y provisto por *Twitter*. Este campo es la clave primaria de la tabla *Tweet* y se almacena como un entero de 8 bytes.
- **text**: Es el texto publicado en el *tweet*. La interfaz gráfica limita su extensión a 140 caracteres. Esta información es susceptible de ser indexada para identificar patrones de similitud entre los usuarios del conjunto de datos. Se almacena como un texto de longitud variable con reserva inicial de 200 caracteres ya que algunos *tweets* pueden exceder los 140 caracteres al incluir referencias a páginas web o fotografías.
- **date**: Fecha de publicación del *Tweet*. Este campo se genera en el momento en el que el usuario publica el *Tweet* y se almacena en la tabla *Tweet* como un entero de 8 bytes representando los milisegundos transcurridos desde 1970-01-01 00:00:00 GMT.
- **favorite_count**: Contador que indica el número de usuarios de *Twitter* que han marcado el *tweet* como favorito hasta el momento de su inserción en la base de datos. Este contador se almacena en la tabla como un entero no negativo y es indicativo de la relevancia del *tweet*.
- **retweet_count**: Contador que indica el número de usuarios de *Twitter* que han *retwitteado* el *tweet* como hasta el momento de su inserción en la base de datos. Este contador se almacena en la tabla como un entero no negativo y es indicativo de la relevancia del *tweet*.

Parte de la información asociada a los tweets como es el caso de los *Hashtags* y las URL citadas pueden tomar múltiples valores dentro de un mismo *tweet*, por este motivo se almacenan en las tablas *TweetComponentHashtag* y *TweetComponentUrl* con la siguiente estructura:

2.1.2.1 TweetComponentHashtag

La tabla *TweetComponentHashtag* almacena los *hashtags* asociados a los *tweets* de la tabla *Tweet*. Aparte del campo **tweet_id** necesario para establecer la relación $1 - n$ con la tabla *Tweet* cuenta con el siguiente campo:

- **hashtag**: Cadena de texto con la que el usuario autor del *tweet* etiqueta al *tweet* contextualmente. Este contenido se almacena en la tabla *TweetComponentHashtag* como un texto de tamaño variable.

2.1.2.2 TweetComponentUrl

La tabla *TweetComponentUrl* almacena las URL asociados a los *tweets* de la tabla *Tweet*. Aparte del campo **tweet_id** necesario para establecer la relación $1 - n$ con la tabla *Tweet* cuenta con los siguientes campos:

- **url**: URL comprimida por *Twitter* que dirige a la URL original asociada al *tweet*. Este contenido se almacena en la tabla *TweetComponentURL* como un texto de tamaño variable. En la propia documentación de la API REST de Twitter [https://dev.twitter.com/docs/platform-objects/entities] se ilustra esto con el ejemplo: `"http:\/\/t.co\/IOwBrTZR"`.
- **expanded_url**: URL original adjunta al *tweet*. Este contenido se almacena en la tabla *TweetComponentURL* como un texto de tamaño variable. En la propia documentación de la API REST de Twitter [https://dev.twitter.com/docs/platform-objects/entities] se ilustra esto con el ejemplo: `"http:\/\/www.youtube.com\/watch?v=oHg5SJYRHA0"`.
- **display_url**: Hipervínculo mostrado en el texto del *tweet* para acceder a la URL original asociada al *tweet*. Este contenido se almacena en la tabla *TweetComponentURL* como un texto de tamaño variable. En la propia documentación de la API REST de Twitter [https://dev.twitter.com/docs/platform-objects/entities] se ilustra esto con el ejemplo: `"youtube.com\/watch?v=oHg5SJ\u2026"`.

2.2 Relaciones del conjunto de datos

2.2.1 SocialLinkDated

La relación *SocialLinkDated* es una relación $m - n$ entre usuarios del conjunto de datos que representa los grafos dirigidos de red social derivados de las acciones *mention*, *reply* y *retweet* en los tweets publicados por los usuarios. Esta relación añade a la arista entre dos usuarios del grafo dirigido dos atributos:

- **type**: Marcador enumerado del tipo de interacción entre los dos usuarios relacionados. Se almacena como una referencia de clave foránea a la tabla de tipos de relaciones entre usuarios *SocialLinkType*. Los tipos que podemos encontrar en la relación *SocialLinkDated* son:
    1) **MENTION**: La arista en el grafo dirigido de red social se corresponde con un *tweet* publicado por el usuario origen de la arista en el que se menciona al usuario destino de la arista.
    2) **REPLY**: La arista en el grafo dirigido de red social se corresponde con un *tweet* publicado por el usuario origen de la arista en respuesta a un *tweet* publicado por el usuario destino de la arista.
    3) **RETWEET**: La arista en el grafo dirigido de red social se corresponde con un *tweet* publicado por el usuario destino de la arista que ha sido *retweeteado* por el usuario origen de la arista.
- **date**: Fecha de publicación del *tweet* o *retweet* en el que se produce la interacción. Este campo se almacena como un entero de 8 bytes representando los milisegundos transcurridos desde 1970-01-01 00:00:00 GMT.

2.2.2 SocialLinkWeighted

La relación *SocialLinkWeighted* es una relación $m - n$ entre usuarios del conjunto de datos que representa el grafo dirigido de red social derivado de las acciones *follow* entre los usuarios del conjunto de datos así como de forma redundante los grafos dirigidos de red social derivados de las acciones *mention*, *reply* y *retweet*.en los tweets publicados por los usuarios. Esta relación añade a la arista entre dos usuarios del grafo dirigido dos atributos:

- **type**: Marcador enumerado del tipo de interacción entre los dos usuarios relacionados. Se almacena como una referencia de clave foránea a la tabla de tipos de relaciones entre usuarios *SocialLinkType*. Los tipos que podemos encontrar en la relación *SocialLinkWeighted* son los mismos que los que encontrábamos en la relación *SocialLinkDated* y además el tipo *FOLLOW*:
    0) **FOLLOW**: La arista en el grafo dirigido de red social indica que el usuario origen de la arista sigue (*follow*) al usuario destino de la arista.

  El hecho de que este tipo no aparezca en la relación anterior se debe a que la API REST de *Twitter* no ofrece información temporal respecto a las acciones *follow* que realizan los usuarios.
- **weight**: Cantidad de repeticiones de la interacción de tipo **type** entre el usuario origen de la arista y el usuario destino de la arista. En particular para interacciones de tipo *FOLLOW* el peso va a ser siempre 1 debido a la naturaleza binaria de este tipo de relación social (un usuario puede seguir o no seguir a otro usuario). Para el resto de interacciones este valor representa la cantidad de veces que se repiten, por ejemplo, la cantidad de veces que un usuario ha *retwitteado* a otro usuario.

2.2.3 Mentions

La relación *Mentions* es una relación $1 - n$ entre un tweet y n posibles usuarios. Esta relación se establece cuando el usuario que publica un tweet incluye en el texto del mismo el *screenname* del usuario mencionado precedido por el caracter especial '@'.

2.2.4 Replies

La relación *Replies* es una relación ternaria entre un *tweet*, el *tweet* al que responde el *tweet* anterior y el usuario que publicó el *tweet* que es respondido. Esta relación se establece cuando el usuario que publica un tweet lo hace en respuesta a un tweet previamente publicado.

2.2.5 Retweets

La relación *Retweets* es una relación ternaria entre un *tweet*, el *retweet* que replica el *tweet* anterior y el usuario que publicó el *tweet* que es replicado o *retwitteado*. Esta relación se establece cuando un usuario *retwittea* un *tweet* previamente publicado.

2.2.6 Composes

La relación *Composes* es la relación $1 - n$ de autoría que relaciona un usuario con los distintos *tweets* que ha publicado.

2.2.7 ExpansionParent

La relación *ExpansionParent* es la relación $1 - n$ que relaciona a un usuario rama o al usuario raíz del conjunto de datos con los usuarios finales que se han incluido en el mismo expandiendo este usuario. Esto permite analizar cuál ha sido el proceso de creación del conjunto de datos una vez éste ha sido creado.

2.3 Tabla Settings

La tabla *Settings* recoge la configuración de *crawling* del conjunto de datos. Es una tabla atípica en el sentido de que sólo va a contener una fila, no obstante, la información contenida en esta fila es necesaria para coordinar distintas instancias del programa *Crawler* colaborando durante la construcción del conjunto de datos.

Todos los campos de esta tabla son parámetros configurables en la ejecución de la primera instancia que se ejecute del programa *Crawler* para construir el conjunto de datos (se pueden consultar los mismos en el anexo 3). Esta instancia escribirá sus parámetros de configuración en la primera fila de la tabla *Settings* y las demás instancias que se adosen a la construcción del conjunto de datos leerán esta tabla para ajustar sus propios parámetros de configuración.

En particular se contemplan dos tipos de parámetros de configuración en esta tabla: *acceptance* y *expansion* diferenciados por el sufijo del nombre del campo en la tabla *Settings*.

Los parámetros *acceptance* determinan requisitos que deben satisfacer, en el momento de la solicitud de su información a la API REST de *Twitter,* los usuarios susceptibles de ser incluidos en el conjunto de datos para ser insertados en el conjunto de datos como usuarios finales del mismo:

- **acceptance_min_tweets_per_user**: Indica el número mínimo de *tweets* que debe haber publicado un usuario hasta el momento de la solicitud de su información a la API REST de *Twitter* para ser insertado en el conjunto de datos como usuario final del mismo. Este requisito sirve para descartar usuarios demasiado inactivos para los cuales la recomendación basada en contenido no dispondría de información suficiente para generar recomendaciones. Se puede omitir este filtro simplemente configurando el programa Crawler inicial estableciendo el valor 0 en el argumento de ejecución `acceptanceMinTweetsPerUser`. En la construcción del conjunto de datos que hemos utilizado en nuestros experimentos el valor por el que hemos optado para la configuración de este campo es 100.
- **acceptance_min_followees_per_user**: Indica el número mínimo de *followees* que debe tener un usuario en el momento de la solicitud de su información a la API REST de *Twitter* para ser insertado en el conjunto de datos como usuario final del mismo. Este requisito sirve para descartar usuarios demasiado inactivos en la red social. Se puede omitir este filtro simplemente configurando el programa Crawler inicial estableciendo el valor 0 en el argumento de ejecución `acceptanceMinFolloweesPerUser`.

En la construcción del conjunto de datos que hemos utilizado en nuestros experimentos el valor por el que hemos optado para la configuración de este campo es 1.

- **acceptance_max_followees_per_user**: Indica el número máximo de *followees* que debe tener un usuario en el momento de la solicitud de su información a la API REST de *Twitter* para ser insertado en el conjunto de datos como usuario final del mismo. Este requisito sirve para descartar cuentas potencialmente robotizadas que puedan distorsionar el conjunto de datos. En la construcción del conjunto de datos que hemos utilizado en nuestros experimentos el valor por el que hemos optado para la configuración de este campo es 2000.
- **acceptance_min_time_since_user_creation**: Indica el tiempo mínimo (en milisegundos) que debe haber transcurrido entre la creación de un usuario y la solicitud de su información a la API REST de *Twitter* para insertar al usuario en el conjunto de datos usuario final del mismo. Este requisito sirve para evitar la inserción de usuarios con un historial excesivamente corto en *Twitter*. Se puede omitir este filtro simplemente configurando el programa Crawler inicial estableciendo el valor 0 en el argumento de ejecución `acceptanceMinTimeSinceUserCreation`. En la construcción del conjunto de datos que hemos utilizado en nuestros experimentos el valor por el que hemos optado para la configuración de este campo es $1000 \cdot 60 \cdot 60 \cdot 24 \cdot 30 \cdot 3$, es decir, hemos insertado como usuarios finales usuarios con al menos tres meses de antigüedad en *Twitter*.

Los parámetros *expansion* configuran el modo de expansión de las instancias de *Crawler* así como el tamaño mínimo del conjunto de datos objetivo de los *Crawler* y consecuentemente la condición de parada del proceso de expansión de usuarios:

- **expansion_number_of_users_to_collect**: Indica el número mínimo de usuarios finales (aquellos han sido insertados en el conjunto de datos por expansión de usuarios rama o del usuario raíz y superan todos los filtros *acceptance*) que debe contener el conjunto de datos para finalizar el proceso de expansión de usuarios rama del conjunto de datos. El número final de usuarios del conjunto de datos será como mínimo igual a `expansion_number_of_users_to_collect`. En la construcción del conjunto de datos que hemos utilizado en nuestros experimentos el valor por el que hemos optado para la configuración de este campo es 10000.
- **expansion_max_number_of_tweets_to_retrieve_per_user**: Indica el número máximo de *tweets* que se recopilan del *timeline* en *Twitter* reciente de cada usuario final del conjunto de datos. En la construcción del conjunto de datos que hemos utilizado en nuestros experimentos el valor por el que hemos optado para la configuración de este campo es 200.
- **expansion_max_number_of_sons_to_expand_per_user**: Indica el número máximo de usuarios seguidos por un usuario rama que pasen los filtros *acceptance* que van a ser insertados en el conjunto de datos como usuarios rama. La configuración de este campo influye considerablemente en la topología del grafo de red social correspondiente a la interacción seguir que presente el conjunto de datos final: un valor 1 producirá el resultado de realizar una búsqueda en profundidad en Twitter y consecuentemente un grafo con múltiples componentes muy separadas (los vecindarios de

los distintos usuarios expandidos) mientras que un valor de -1 (que es el valor provisto para considerar a todos los usuarios del conjunto de datos como usuarios rama) producirá el resultado de realizar una búsqueda en anchura en Twitter y consecuentemente un grafo con una gran componente altamente homogénea y con menor clustering. En la construcción del conjunto de datos que hemos utilizado en nuestros experimentos el valor por el que hemos optado para la configuración de este campo es 5. Esto nos ofrece una solución intermedia entre los dos extremos presentados dando lugar a un subconjunto de *Twitter* cuya red social es más similar a la red total de *Twitter*.

- **expansion_root_user_id**: Se trata del identificador del usuario raíz a partir del cual se inicial la construcción del conjunto de datos. En la construcción del conjunto de datos que hemos utilizado en nuestros experimentos el valor por el que hemos optado para la configuración de este campo es el identificador de un usuario elegido al azar que supera los filtros *acceptance* presentados con anterioridad: 38292395.

3. Anexo III: Salida de ejecución del programa Help

En este anexo se presenta la salida de la ejecución del programa *Help* de la plataforma de experimentación desarrollada. Esta salida presenta todos los ejecutables desarrollados junto con todos sus parámetros de configuración y el método de utilización de los mismos así como sus valores por defecto.

```
This twitter toolset has different executable programs, to execute one of them, its class must be specified in the first execution argument.
  Currently executable programs are:
    es.uam.eps.ir.twitter.executable.help.Help
    es.uam.eps.ir.twitter.executable.crawler.Crawler
    es.uam.eps.ir.twitter.executable.dbisolator.DBIsolator
    es.uam.eps.ir.twitter.executable.graphgen.Graphgen
    es.uam.eps.ir.twitter.executable.dataset2zip.Dataset2Zip
    es.uam.eps.ir.twitter.executable.splitter.Splitter
    es.uam.eps.ir.twitter.executable.indexing.Indexer
    es.uam.eps.ir.twitter.executable.indexing.InteractionMapBuilder
    es.uam.eps.ir.twitter.executable.indexing.CentroidBuilder
    es.uam.eps.ir.twitter.executable.recommendations.RecommendationsGenerator
    es.uam.eps.ir.twitter.executable.evaluations.Evaluator
Every one of these programs accept optional arguments which work the following way:
Execution arguments are optional:
  Accepted arguments follow the next format:
    [-<SettingsField> <valueToAsign>] [-<SettingsField> <valueToAsign>] ...
    Where:
      > SettingsField is a Settings modifiable field (a public setter exists and can be found in the app javadoc).
      > valueToAssign is the value to be assigned to the preceding field.
    Note that for time related suffix (In[Milliseconds/Seconds/Minutes/Hours/Days/Months/Years]) can be added
    to determinate the way of parsing the assigned value.
      Common settings:
      +-----------------------------------------------------+-----------------------------------+-----------------------------------------------------------------------------+
      | SettingsField                                       | ValueToAsign type                 | Accepted time related suffix / Accepted enum values                         |
      +-----------------------------------------------------+-----------------------------------+-----------------------------------------------------------------------------+
      | twitter4jMaxTwitterTryCount                         | Integer                           |                                                                             |
      | executionWaitTimeForOtherThreadsToWork              | Integer                           |                                                                             |
      | executionDebugging                                  | Boolean                           |                                                                             |
      | executionTrecEvalProgramPath                        | String                            |                                                                             |
      +.....................................................+...................................+.............................................................................+
      | serializedDataSocialLinkType                        | Enum description String           | Follow                                                                      |
      |                                                     |                                   | Mention                                                                     |
      |                                                     |                                   | Reply                                                                       |
      |                                                     |                                   | Retweet                                                                     |
      +.....................................................+...................................+.............................................................................+
      | interactionMapBuildingAlgorithmChoice               | EnumInteractionMapBuildingAlgorithm | ConstantForSelfWrittenTweets                                              |
      |                                                     |                                   | TimeLDependantForSelfWrittenTweets                                          |
      +-----------------------------------------------------+-----------------------------------+-----------------------------------------------------------------------------+
```

Crawler specific settings:

SettingsField	ValueToAsign type	Accepted time related suffix / Accepted enum values
acceptanceMinTweetsPerUser	Long	
acceptanceMinFolloweesPerUser	Long	
acceptanceMaxFolloweesPerUser	Long	
acceptanceMinTimeSinceUserCreation	Long	In[Milliseconds/Seconds/Minutes/Hours/Days/Months/Years]
expansionNumberOfUsersToCollect	Long	
expansionMaxNumberOfTweetsToRetrievePerUser	Integer	
expansionMaxNumberOfSonsToExpandPerUser	Long / "all"	
expansionRootUserId	Long	
executionNShowUserThreadsPerGetFriendsIDsThread	Integer	

Graphgen specific settings:

SettingsField	ValueToAsign type	Accepted time related suffix / Accepted enum values
graphgenCsvSeparator	String	

Dataset2Zip specific settings:

SettingsField	ValueToAsign type	Accepted time related suffix / Accepted enum values
dataset2ZipOverwrite	Boolean	

Splitter specific settings:

SettingsField	ValueToAsign type	Accepted time related suffix / Accepted enum values
splitterTrainDatasetProportion	Double	
splitterOverwrite	Boolean	
splitterIntegrityCheck	Boolean	
splitterAlgorithmChoice	EnumTwitterSplitImplementation	Random
		Chronological

Indexer specific settings:

SettingsField	ValueToAsign type	Accepted time related suffix / Accepted enum values
indexerOverwrite	Boolean	

InteractionMapBuilder specific settings:

SettingsField	ValueToAsign type	Accepted time related suffix / Accepted enum values
interactionMapBuilderOverwrite	Boolean	

CentroidBuilder specific settings:

SettingsField	ValueToAsign type	Accepted time related suffix / Accepted enum values
centroidBuilderOverwrite	Boolean	

RecommendationsGenerator specific settings:

SettingsField	ValueToAsign type	Accepted time related suffix / Accepted enum values
recommendationsGeneratorCutoff	Integer / "all"	
recommendationsGeneratorOverwrite	Boolean	
recommendationAlgorithm	Regular Expression String	Random[_<argument>=<value/s>[/<arument>=<value/s>[/...]]}] MaxFOAF[_<argument>=<value/s>[/<arument>=<value/s>[/...]]}] Popularity[_<argument>=<value/s>[/<arument>=<value/s>[/...]]}] PageRank[_<argument>=<value/s>[/<arument>=<value/s>[/...]]}] PersonalizedPageRank[_<argument>=<value/s>[/<arument>=<value/s>[/...]]}] HITSAuthority[_<argument>=<value/s>[/<arument>=<value/s>[/...]]}] HITSHub[_<argument>=<value/s>[/<arument>=<value/s>[/...]]}] BetwennessCentrality[_<argument>=<value/s>[/<arument>=<value/s>[/...]]}] Rocchio[_<argument>=<value/s>[/<arument>=<value/s>[/...]]}] Hybrid[_<argument>=<value/s>[/<arument>=<value/s>[/...]]}]
recommendationsGeneratorNumRecommendationThreads	Integer	

Regular Expression String argument values can be represented in three different ways:
- Single value:..... <value>
- List of values:..... <value>,<value>[,<value>[,...]]
- Range of numerical values:... <first_value>:<last_value>[:<increment>]
  (If not specified, increment will be 1)

Evaluator specific settings:

SettingsField	ValueToAsign type	Accepted time related suffix / Accepted enum values
evaluatorOverwrite	Boolean	
evaluatorCutoff	Integer	(Multiple separatedly declared evaluatorCutoffs are supported)
evaluatorGenerateEvaluationsPerUser	Boolean	
evaluatorRecommendationAlgorithm	Regular Expression String	Random[_<argument>=<value/s>[/<arument>=<value/s>[/...]]}] MaxFOAF[_<argument>=<value/s>[/<arument>=<value/s>[/...]]}] Popularity[_<argument>=<value/s>[/<arument>=<value/s>[/...]]}] PageRank[_<argument>=<value/s>[/<arument>=<value/s>[/...]]}] PersonalizedPageRank[_<argument>=<value/s>[/<arument>=<value/s>[/...]]}] HITSAuthority[_<argument>=<value/s>[/<arument>=<value/s>[/...]]}] HITSHub[_<argument>=<value/s>[/<arument>=<value/s>[/...]]}] BetwennessCentrality[_<argument>=<value/s>[/<arument>=<value/s>[/...]]}] Rocchio[_<argument>=<value/s>[/<arument>=<value/s>[/...]]}] Hybrid[_<argument>=<value/s>[/<arument>=<value/s>[/...]]}]

```
+-------------------------------------------------+-----------------------------------+-------------------------------------------------------------------------+
| evaluatorTrecUseTrec                            | Boolean                           |                                                                         |
| evaluatorTrecThreshold                          | Integer                           | (Multiple separatedly declared evaluatorTrecThresholds are supported)   |
+-------------------------------------------------+-----------------------------------+-------------------------------------------------------------------------+
| evaluatorTrecMeasure                            | Trec Measure Description String   | Please, consult trec_eval documentation for more information.           |
|                                                 |                                   | (Multiple separatedly declared evaluatorTrecMeasures are supported)     |
+-------------------------------------------------+-----------------------------------+-------------------------------------------------------------------------+
| evaluatorTrecUseAllUrelsFileValues              | Boolean                           |                                                                         |
+-------------------------------------------------+-----------------------------------+-------------------------------------------------------------------------+

  Regular Expression String argument values can be represented in three different ways:
     Single value:................ <value>
     List of values:.............. <value>,<value>[,<value>[,...]]
     Range of numerical values:... <first_value>:<last_value>[:<increment>]
                                                    (If not specified, increment will be 1)

Now all configurable settings values are going to be restored to their default values and displayed.
Please, take into account that all directory settings containing the word "Help" or "Unknown"
would be substituted by the executable class name when executing one of the executable classes:
=============================================================================================================================================================================
  Twitter Help settings configuration after calling es.uam.eps.ir.twitter.common.settings.Settings.restoreDefaultSettings:
    Common settings:
      Twitter4j settings:
        twitter4jMaxTwitterTryCount:.....................5
      Execution mode settings:
        executionDebugging:..............................false
      DAO settings:
        daoPropertiesCrawlerSpecificKey:.................twitter.crawler.jdbc
        daoPropertiesIsolatedSpecificKey:................twitter.isolated.jdbc
      Execution location:
        executionLocation:...............................galactica.ii.uam.es
        executionExecutingInExecutionServer:.............true
      Execution performance settings:
        executionWaitTimeForOtherThreadsToWork:..........60
      TREC settings:
        executionTrecEvalProgramPath:....................../home/alejandro/trec_eval.9.0/trec_eval
      Serialized data settings:
        serializedDataSocialLinkTypes:...................Follow
        serializedDataContainingTimeInformation:.........false
      Interaction map building settings:
        interactionMapBuildingAlgorithmChoice:...........ConstantForSelfWrittenTweets
      Directory settings:
        directoryCollectionsDir:.........................../home/guest/alfonso/collections/
        directoryDatasetsDir:............................/home/guest/alfonso/collections/datasets/
        directoryResDir:.................................../home/guest/alfonso/collections/alfonso/application_resources/twitter/res/
        directoryJsonDir:................................/home/guest/alfonso/collections/alfonso/application_resources/twitter/res/JSON/
        directoryJsonStatusDir:............................/home/guest/alfonso/collections/alfonso/application_resources/twitter/res/JSON/Status/
        directoryJsonUserDir:............................/home/guest/alfonso/collections/alfonso/application_resources/twitter/res/JSON/User/
        directoryJsonFollowDir:............................/home/guest/alfonso/collections/alfonso/application_resources/twitter/res/JSON/Follow/
        directoryLogDir:.................................../home/guest/alfonso/collections/alfonso/application_resources/twitter/res/log/
        directoryLogSpecificDir:.........................../home/guest/alfonso/collections/alfonso/application_resources/twitter/res/log/Help/
        directoryCsvDir:.................................../home/guest/alfonso/collections/alfonso/application_resources/twitter/res/csv/
```

```
            directoryCsvSpecificDir:........................../home/guest/alfonso/collections/alfonso/application_resources/twitter/res/csv/
            directorySerializedDataDir:........................../home/guest/alfonso/collections/alfonso/application_resources/twitter/res/data/
            directorySerializedDataSpecificDir:................../home/guest/alfonso/collections/alfonso/application_resources/twitter/res/data/alfonso-twitter-isolated/Follow/
            directoryRecsDir:..................................../home/guest/alfonso/collections/alfonso/application_resources/twitter/res/recs/
            directoryRecsSpecificDir:............................/home/guest/alfonso/collections/alfonso/application_resources/twitter/res/recs/alfonso-twitter-isolated/Follow/
            directoryUrelsDir:.................................../home/guest/alfonso/collections/alfonso/application_resources/twitter/res/urels/
            directoryUrelsSpecificDir:......................../home/guest/alfonso/collections/alfonso/application_resources/twitter/res/urels/alfonso-twitter-isolated/Follow/
            directoryEvaluationsDir:........................../home/guest/alfonso/collections/alfonso/application_resources/twitter/res/evaluations/
            directoryEvaluationsSpecificDir:................../home/guest/alfonso/collections/alfonso/application_resources/twitter/res/evaluations/alfonso-twitter-isolated/Follow/
            directoryIndexDir:.................................../home/guest/alfonso/collections/alfonso/application_resources/twitter/res/index/
            directoryIndexTweetsIndexDir:......................../home/guest/alfonso/collections/alfonso/application_resources/twitter/res/index/tweets_index/
            directoryIndexTweetsIndexSpecificDir:............../home/guest/alfonso/collections/alfonso/application_resources/twitter/res/index/tweets_index/alfonso-twitter-iso-
lated/Follow/
            directoryPathSeparator:............................/
          Crawler specific settings:
            Execution performance settings:
              executionNShowUserThreadsPerGetFriendsIDsThread:...20
            User acceptance settings:
              acceptanceMinTweetsPerUser:........................100
              acceptanceMinFolloweesPerUser:.....................1
              acceptanceMaxFolloweesPerUser:.....................2000
              acceptanceMinTimeSinceUserCreation:.................90 days, 0 hours, 0 minutes, 0 seconds and 0 milliseconds (7776000000 milliseconds)
            Graph expansion settings:
              expansionNumberOfUsersToCollect:...................10000
              expansionMaxNumberOfTweetsToRetrievePerUser:.......200
              expansionMaxNumberOfSonsToExpandPerUser:...........5
              expansionRootUserId:...............................38292395
          Graphgen specific settings:
            Csv settings:
              graphgenCsvSeparator:..............................','
          Dataset2Zip specific settings:
            Overwrite settings:
              dataset2ZipOverwrite:..............................false
          Splitter specific settings:
            Output settings:
              splitterOverwrite:.................................false
            IntegrityCheck:
              splitterIntegrityCheck:............................false
            Splitting settings:
              splitterTrainDatasetProportion:....................0.6
              splitterAlgorithmChoice:...........................Random
          Indexer specific settings:
            Output settings:
              indexerOverwrite:..................................false
          InteractionMapBuilder specific settings:
            Output settings:
              interactionMapBuilderOverwrite:....................false
          CentroidBuilder specific settings:
            Output settings:
              centroidBuilderOverwrite:..........................false
```

```
 RecommendationsGenerator specific settings:
   Output settings:
     recommendationsGeneratorCutoff:.....................1000
     recommendationsGeneratorOverwrite:..................false
   Recommendations generation settings:
     recommendationsGeneratorAlgorithmChoices:..........No algorithms have been chosen yet.
   Performance settings:
     recommendationsGeneratorNumRecommendationThreads:..1
 Evaluator specific settings:
    Output settings:
     evaluatorOverwrite:................................false
    Evaluation settings:
     evaluatorCutoffs:..................................{1000}
     evaluatorGenerateEvaluationsPerUser.................false
     evaluatorRecommendationAlgorithmChoices:...........No algorithms have been chosen yet.
    Evaluation trec settings:
     evaluatorTrecUseTrec:..............................true
     evaluatorTrecThresholds:...........................{1}
     evaluatorTrecMeasures:.............................{}
evaluatorTrecUseAllUrelsFileValues:................false
```

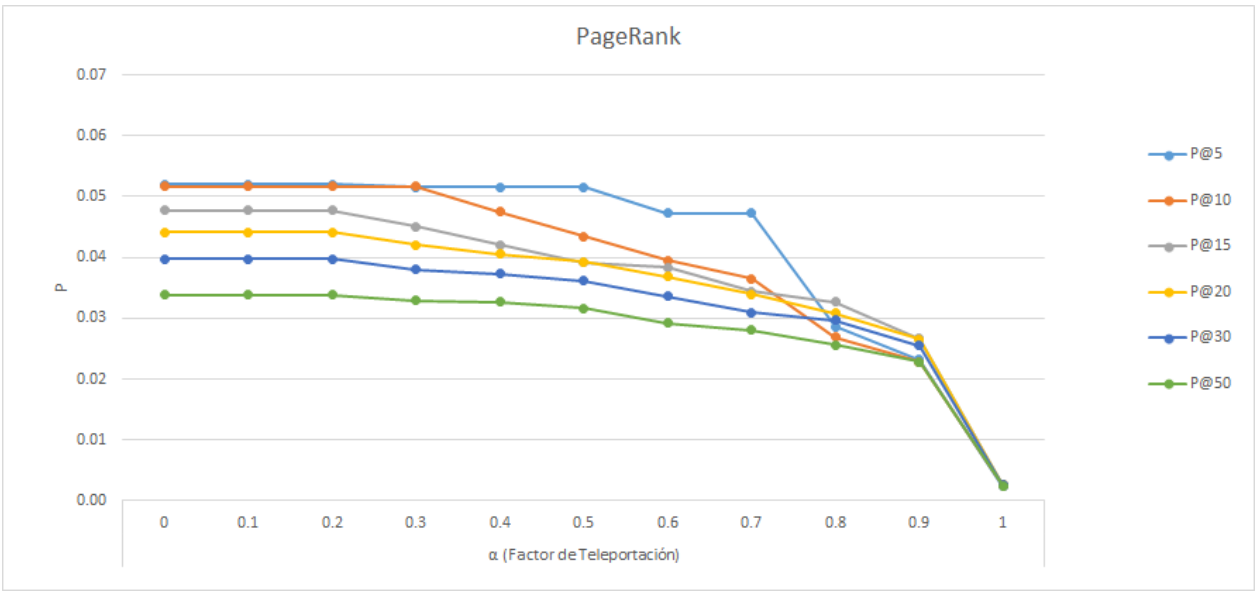
4. Anexo IV: Resultados experimentales en detalle

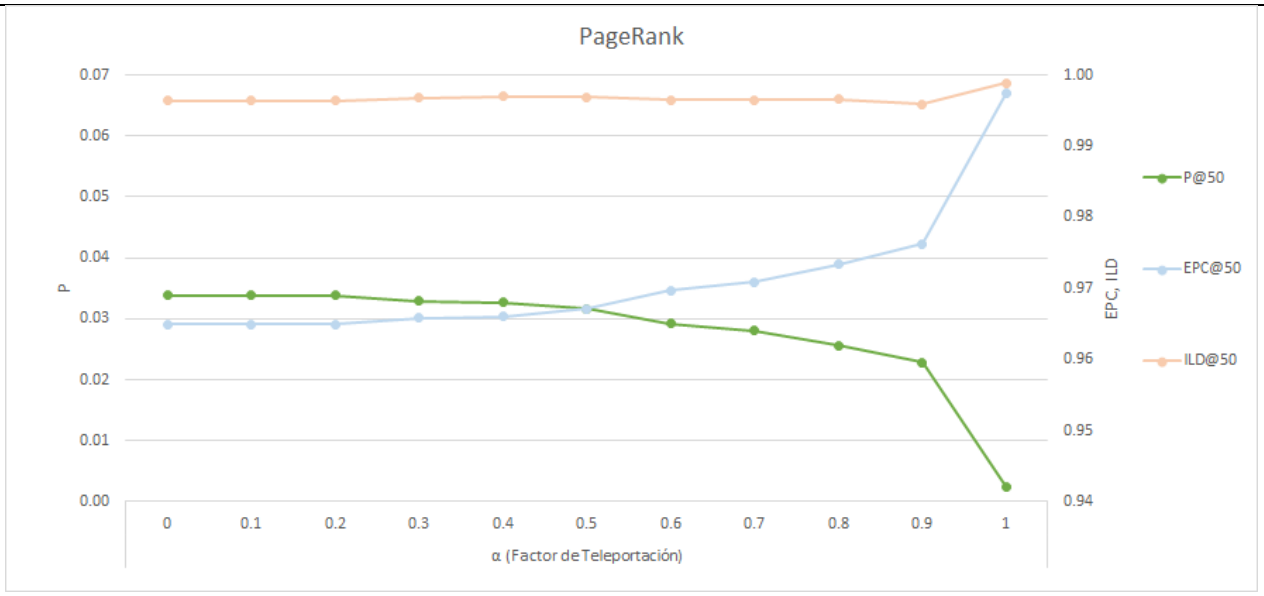
En el presente anexo se presentan las tablas de resultados y las gráficas representativas de los mismos que reflejan el comportamiento de las variaciones de los distintos parámetros configurables de los algoritmos de recomendación que los admiten. Como se puede comprobar, las configuraciones que maximizan la métrica de precisión en *cutoff* 5 son las que han sido seleccionadas para la presentación de los resultados finales de evaluación que se presentan en el apartado 6.2.

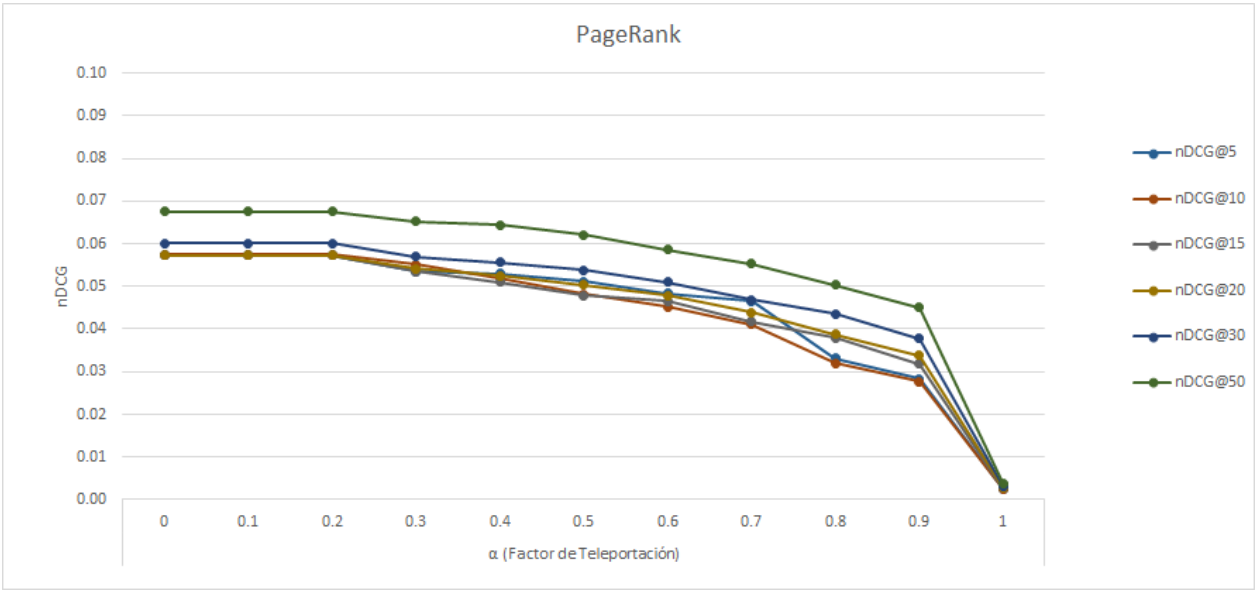
4.1 Resultados en el conjunto de datos *follow*

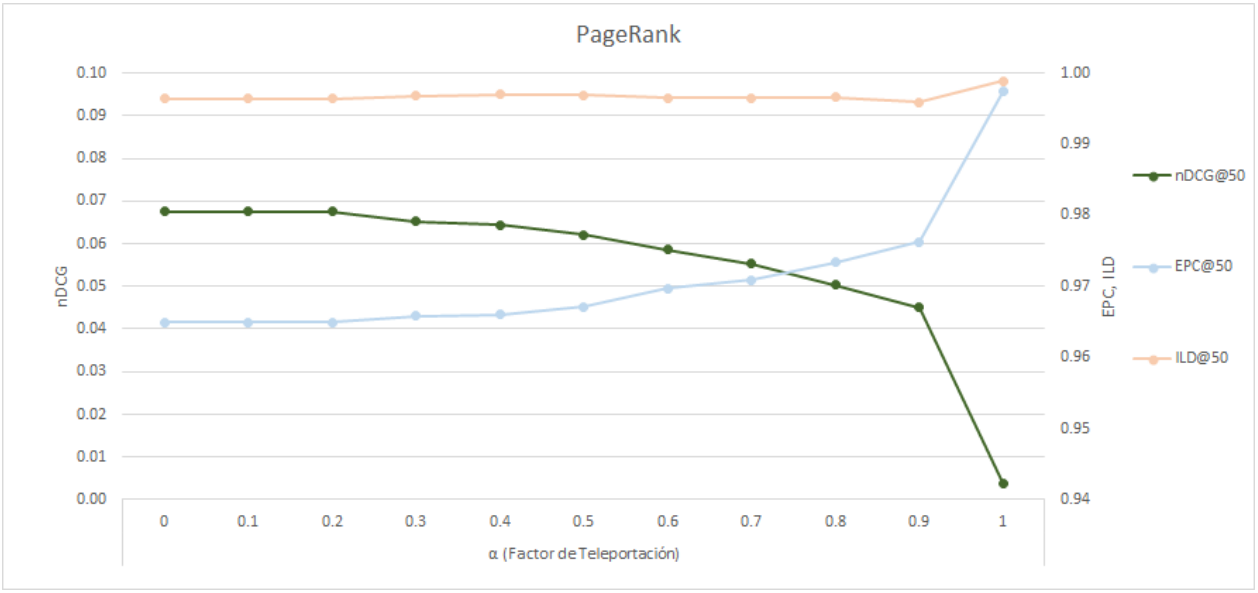
4.1.1 Efecto en las evaluaciones de la variación del factor de teleportación en el algoritmo PageRank

α (Factor de Teleportación)	P@5	P@10	P@15	P@20	P@30	P@50	nDCG@5	nDCG@10	nDCG@15	nDCG@20	nDCG@30	nDCG@50	EPC@50	ILD@50
0	0.0520	0.0517	0.0477	0.0441	0.0397	0.0338	0.0572	0.0575	0.0572	0.0573	0.0601	0.0675	0.9649	0.9964
0.1	0.0520	0.0517	0.0477	0.0441	0.0397	0.0338	0.0572	0.0575	0.0572	0.0573	0.0601	0.0675	0.9649	0.9964
0.2	0.0520	0.0517	0.0477	0.0441	0.0397	0.0338	0.0572	0.0575	0.0572	0.0573	0.0601	0.0675	0.9649	0.9964
0.3	0.0515	0.0517	0.0451	0.0421	0.0380	0.0329	0.0534	0.0552	0.0535	0.0541	0.0570	0.0652	0.9658	0.9968
0.4	0.0515	0.0475	0.0421	0.0405	0.0373	0.0327	0.0530	0.0520	0.0510	0.0525	0.0557	0.0644	0.9660	0.9970
0.5	0.0515	0.0434	0.0392	0.0393	0.0361	0.0316	0.0512	0.0482	0.0478	0.0503	0.0538	0.0621	0.9671	0.9969
0.6	0.0473	0.0395	0.0383	0.0368	0.0336	0.0292	0.0484	0.0452	0.0466	0.0478	0.0510	0.0586	0.9697	0.9965
0.7	0.0473	0.0365	0.0344	0.0340	0.0310	0.0280	0.0466	0.0410	0.0418	0.0439	0.0469	0.0553	0.9709	0.9965
0.8	0.0286	0.0269	0.0327	0.0308	0.0297	0.0256	0.0332	0.0320	0.0379	0.0388	0.0435	0.0503	0.9734	0.9966
0.9	0.0232	0.0228	0.0267	0.0266	0.0255	0.0229	0.0283	0.0277	0.0318	0.0338	0.0377	0.0450	0.9762	0.9959
1	0.0024	0.0026	0.0025	0.0026	0.0026	0.0025	0.0026	0.0026	0.0027	0.0029	0.0032	0.0039	0.9974	0.9989



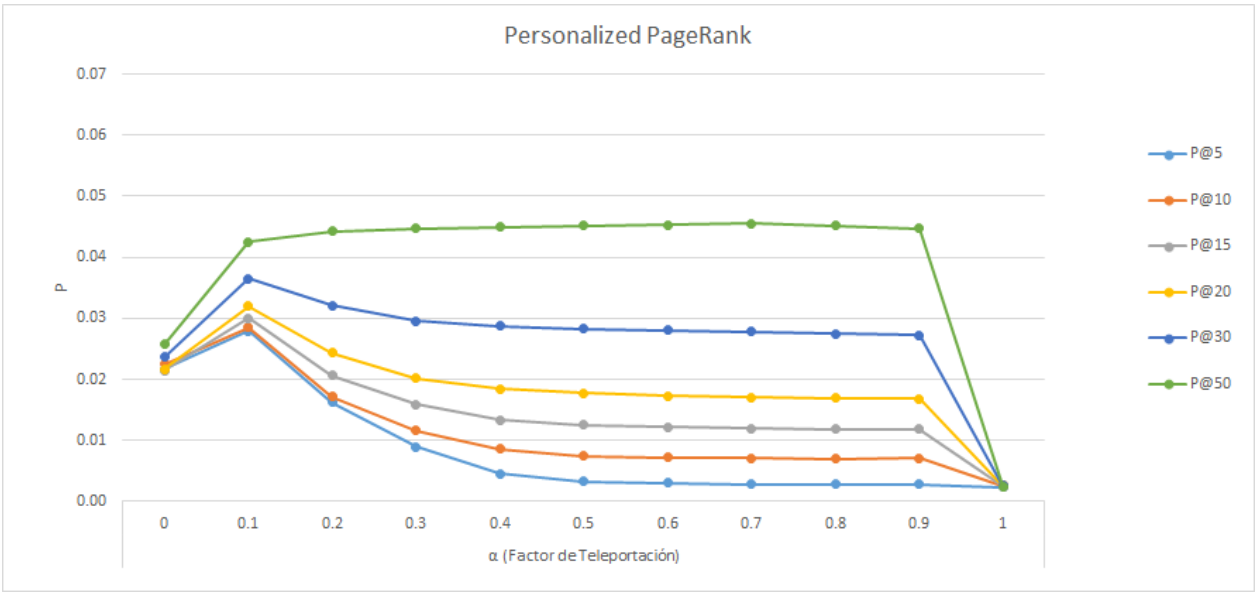

PageRank
P
EPC, ILD
α (Factor de Teleportación)
P@50
EPC@50
ILD@50

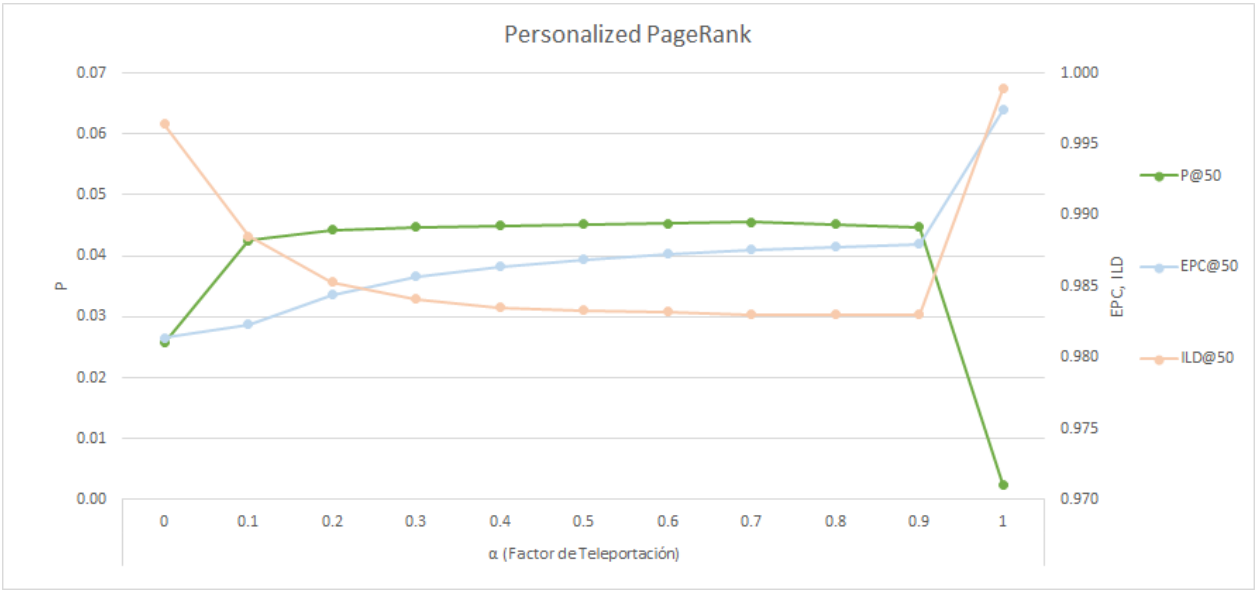

PageRank
nDCG
α (Factor de Teleportación)
nDCG@5
nDCG@10
nDCG@15
nDCG@20
nDCG@30
nDCG@50

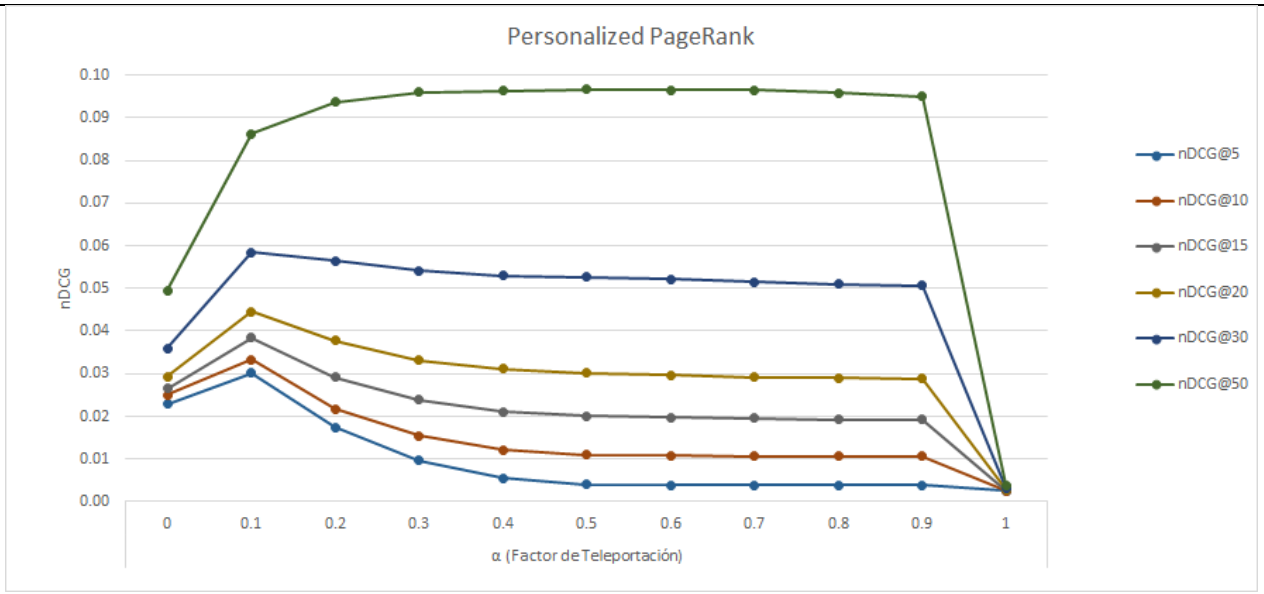

PageRank
nDCG
EPC, ILD
α (Factor de Teleportación)
nDCG@50
EPC@50
ILD@50

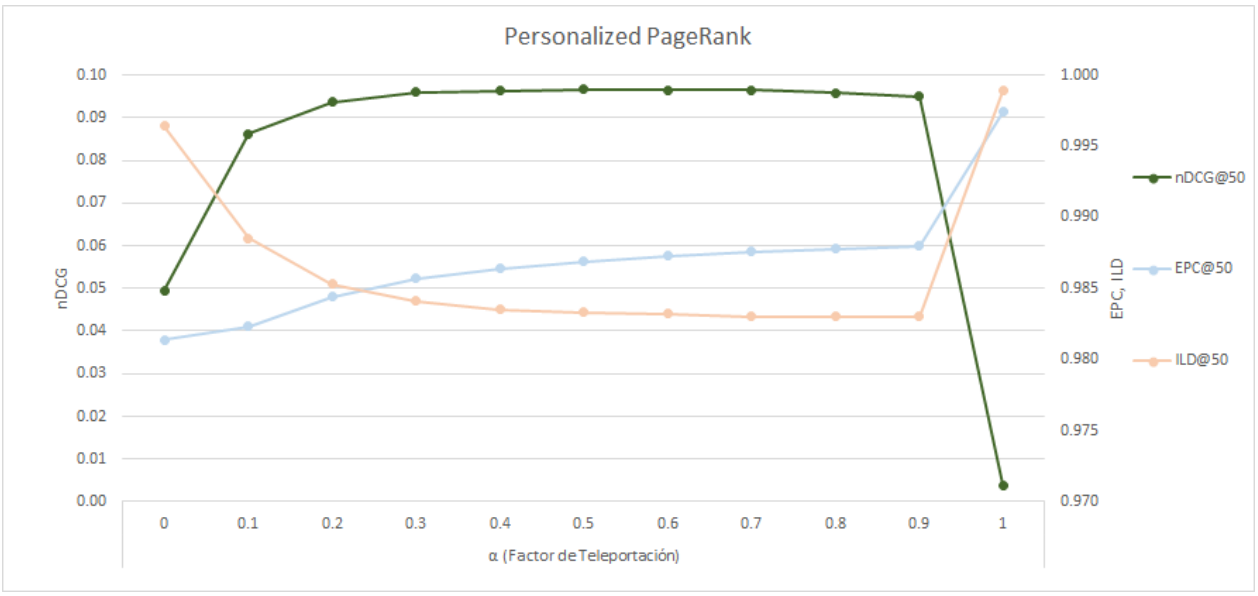
4.1.2 Efecto en las evaluaciones de la variación del factor de teleportación en el algoritmo PageRank Personalizado

α (Factor de Teleportación)	P@5	P@10	P@15	P@20	P@30	P@50	nDCG@5	nDCG@10	nDCG@15	nDCG@20	nDCG@30	nDCG@50	EPC@50	ILD@50
0	0.0217	0.0225	0.0214	0.0217	0.0237	0.0257	0.0228	0.0251	0.0265	0.0293	0.0360	0.0495	0.9814	0.9964
0.1	0.0279	0.0285	0.0300	0.0320	0.0365	0.0425	0.0301	0.0333	0.0385	0.0446	0.0584	0.0862	0.9823	0.9885
0.2	0.0162	0.0172	0.0207	0.0244	0.0321	0.0443	0.0175	0.0218	0.0292	0.0377	0.0565	0.0937	0.9844	0.9853
0.3	0.0089	0.0116	0.0159	0.0202	0.0295	0.0447	0.0097	0.0154	0.0238	0.0332	0.0541	0.0959	0.9857	0.9841
0.4	0.0046	0.0086	0.0133	0.0185	0.0287	0.0450	0.0055	0.0122	0.0210	0.0311	0.0530	0.0963	0.9864	0.9835
0.5	0.0033	0.0075	0.0125	0.0177	0.0283	0.0452	0.0040	0.0110	0.0200	0.0301	0.0526	0.0966	0.9869	0.9833
0.6	0.0030	0.0072	0.0122	0.0173	0.0280	0.0453	0.0038	0.0108	0.0198	0.0297	0.0521	0.0965	0.9873	0.9832
0.7	0.0028	0.0071	0.0120	0.0170	0.0278	0.0455	0.0038	0.0107	0.0195	0.0291	0.0515	0.0965	0.9876	0.9830
0.8	0.0028	0.0070	0.0118	0.0169	0.0275	0.0452	0.0038	0.0106	0.0193	0.0290	0.0510	0.0958	0.9878	0.9830
0.9	0.0028	0.0071	0.0118	0.0168	0.0272	0.0447	0.0038	0.0107	0.0192	0.0288	0.0507	0.0949	0.9880	0.9830
1	0.0024	0.0026	0.0025	0.0026	0.0026	0.0025	0.0026	0.0026	0.0027	0.0029	0.0032	0.0039	0.9974	0.9989



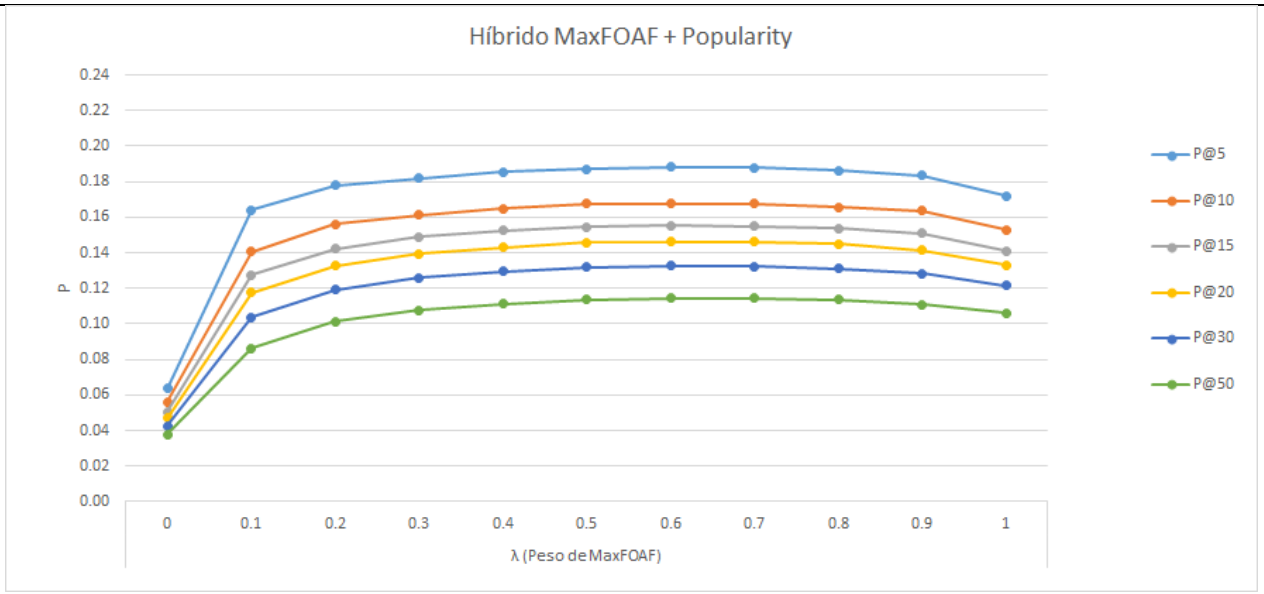


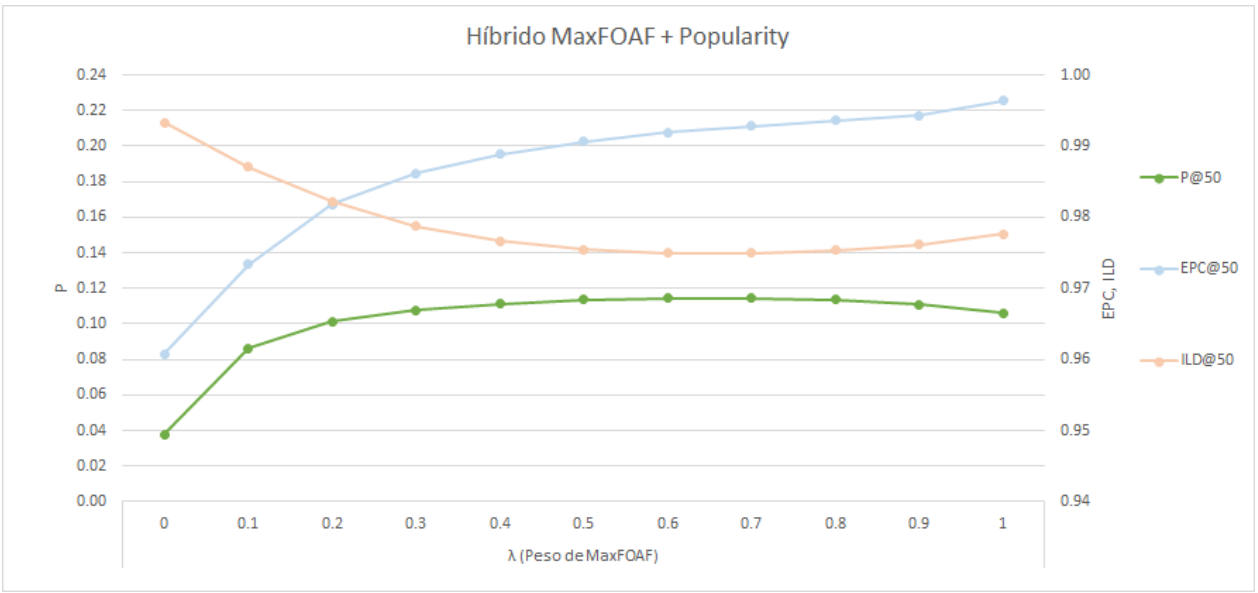


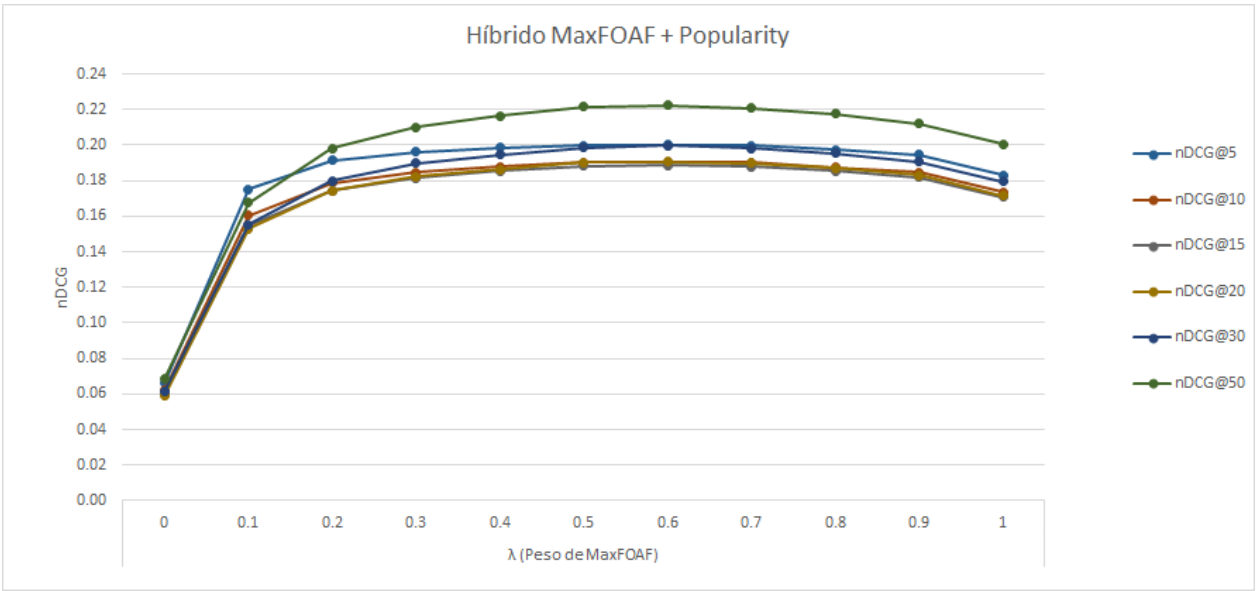


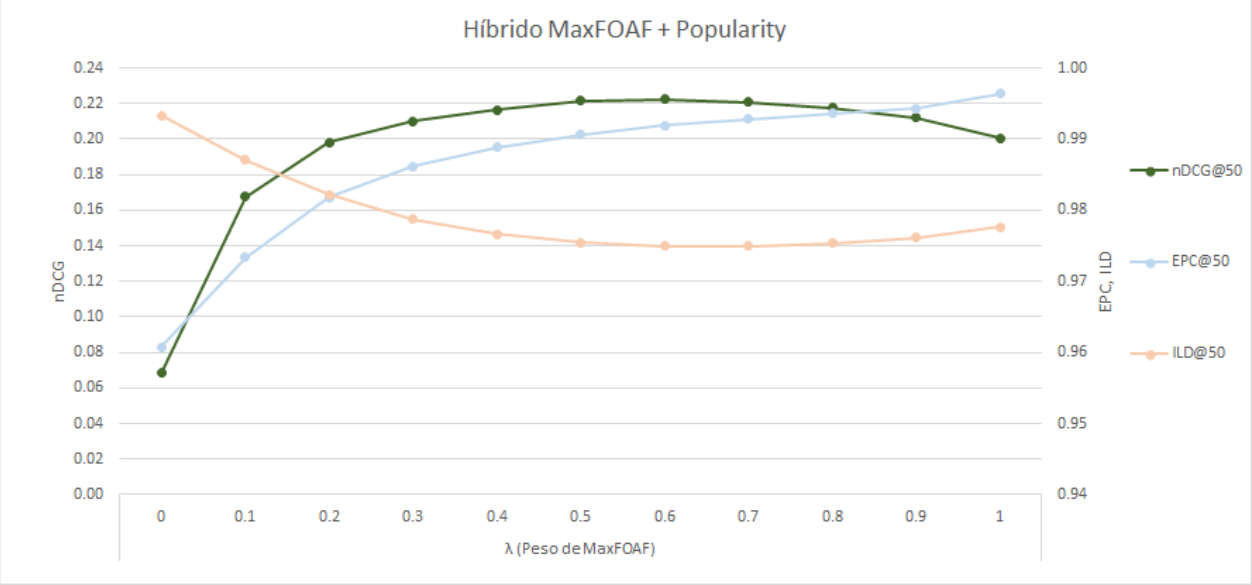
4.1.3 Efecto en las evaluaciones de la variación del peso de cada algoritmo en el algoritmo híbrido MaxFOAF + Popularity

λ (Peso de MaxFOAF)	P@5	P@10	P@15	P@20	P@30	P@50	nDCG@5	nDCG@10	nDCG@15	nDCG@20	nDCG@30	nDCG@50	EPC@50	ILD@50
0	0.0635	0.0561	0.0506	0.0471	0.0428	0.0377	0.0662	0.0619	0.0595	0.0588	0.0614	0.0685	0.9608	0.9933
0.1	0.1640	0.1408	0.1275	0.1178	0.1036	0.0861	0.1751	0.1604	0.1547	0.1531	0.1555	0.1675	0.9734	0.9871
0.2	0.1781	0.1561	0.1422	0.1328	0.1191	0.1015	0.1915	0.1786	0.1742	0.1742	0.1801	0.1980	0.9818	0.9822
0.3	0.1819	0.1614	0.1488	0.1394	0.1258	0.1076	0.1961	0.1847	0.1815	0.1824	0.1897	0.2101	0.9862	0.9787
0.4	0.1855	0.1648	0.1526	0.1431	0.1294	0.1111	0.1987	0.1878	0.1854	0.1864	0.1946	0.2165	0.9888	0.9766
0.5	0.1870	0.1675	0.1545	0.1457	0.1317	0.1136	0.1997	0.1904	0.1881	0.1901	0.1987	0.2217	0.9906	0.9755
0.6	0.1882	0.1676	0.1554	0.1463	0.1328	0.1144	0.2003	0.1903	0.1888	0.1907	0.1996	0.2223	0.9919	0.9750
0.7	0.1879	0.1678	0.1548	0.1462	0.1324	0.1143	0.1999	0.1901	0.1878	0.1900	0.1983	0.2209	0.9928	0.9750
0.8	0.1861	0.1657	0.1537	0.1448	0.1313	0.1135	0.1973	0.1873	0.1854	0.1872	0.1953	0.2176	0.9936	0.9754
0.9	0.1835	0.1637	0.1509	0.1414	0.1284	0.1108	0.1946	0.1847	0.1820	0.1829	0.1908	0.2122	0.9943	0.9761
1	0.1719	0.1529	0.1412	0.1330	0.1216	0.1061	0.1832	0.1734	0.1706	0.1717	0.1794	0.2007	0.9964	0.9776


Híbrido MaxFOAF + Popularity
P
λ (Peso de MaxFOAF)
P@5
P@10
P@15
P@20
P@30
P@50


Híbrido MaxFOAF + Popularity
P
EPC, ILD
λ (Peso de MaxFOAF)
P@50
EPC@50
ILD@50

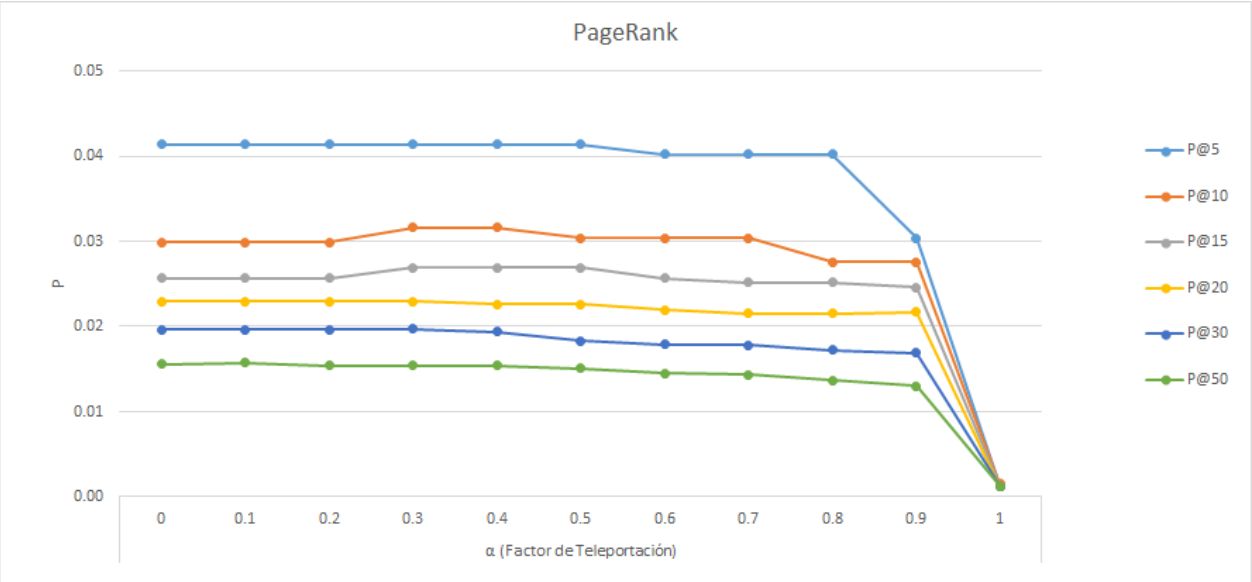

Híbrido MaxFOAF + Popularity
nDCG
λ (Peso de MaxFOAF)
nDCG@5
nDCG@10
nDCG@15
nDCG@20
nDCG@30
nDCG@50

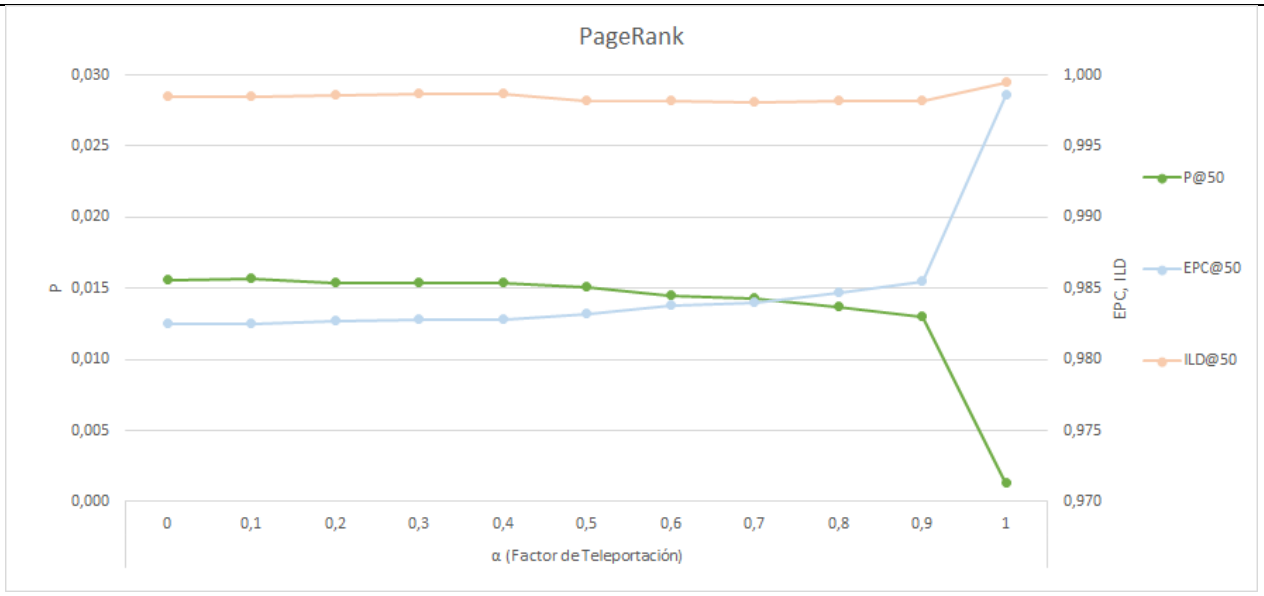


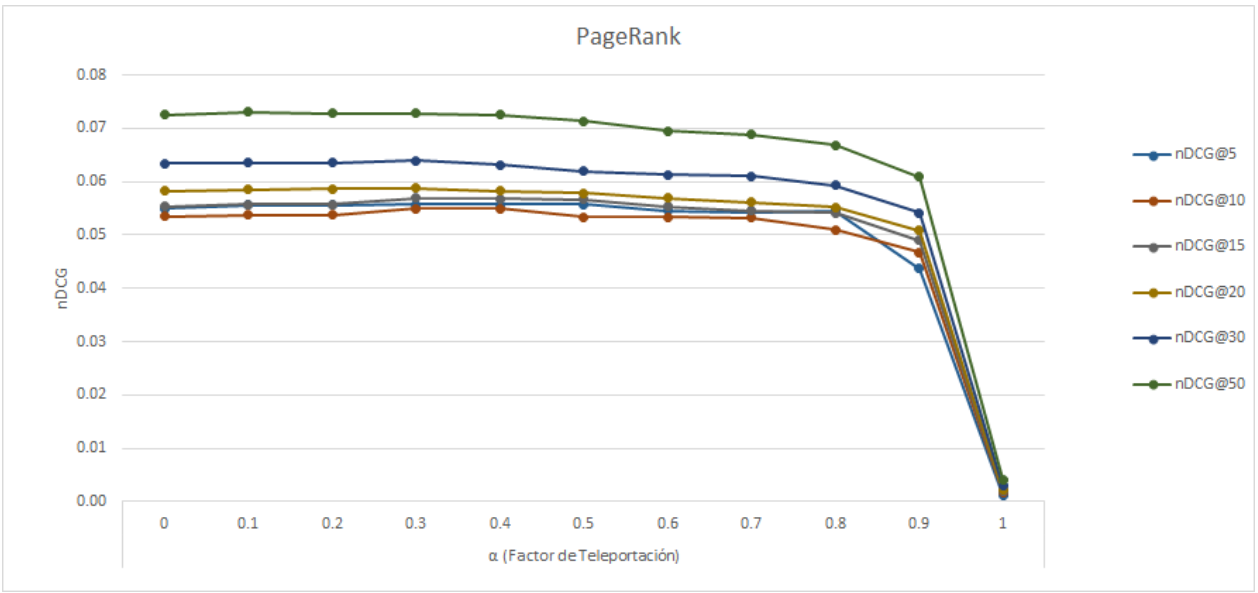
4.2 Resultados en el conjunto de datos *mention / reply / retweet*

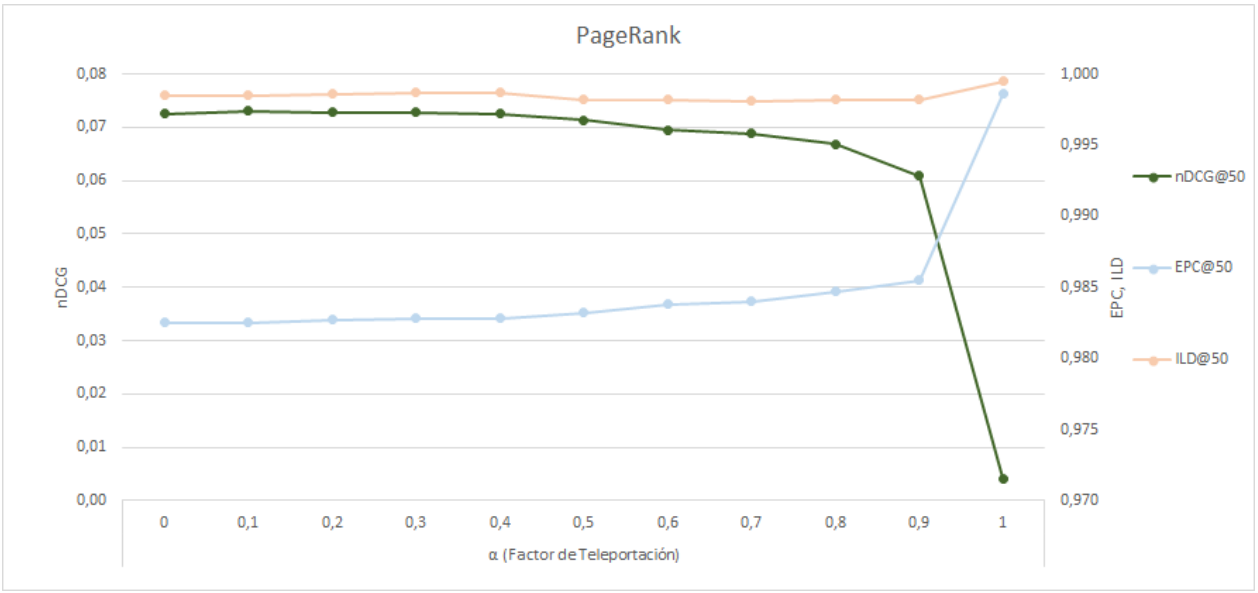
4.2.1 Efecto en las evaluaciones de la variación del factor de teleportación en el algoritmo PageRank

α (Factor de Teleportación)	P@5	P@10	P@15	P@20	P@30	P@50	nDCG@5	nDCG@10	nDCG@15	nDCG@20	nDCG@30	nDCG@50	EPC@50	ILD@50
0	0.0414	0.0299	0.0257	0.0229	0.0196	0.0156	0.0550	0.0535	0.0554	0.0583	0.0634	0.0726	0.9825	0.9985
0.1	0.0414	0.0299	0.0257	0.0229	0.0196	0.0157	0.0556	0.0538	0.0557	0.0585	0.0636	0.0731	0.9825	0.9985
0.2	0.0414	0.0299	0.0257	0.0229	0.0196	0.0154	0.0556	0.0538	0.0557	0.0586	0.0636	0.0728	0.9827	0.9986
0.3	0.0414	0.0316	0.0269	0.0229	0.0197	0.0154	0.0558	0.0549	0.0569	0.0588	0.0639	0.0728	0.9828	0.9987
0.4	0.0414	0.0316	0.0269	0.0226	0.0194	0.0154	0.0558	0.0549	0.0568	0.0582	0.0632	0.0725	0.9828	0.9987
0.5	0.0414	0.0304	0.0269	0.0226	0.0183	0.0151	0.0558	0.0534	0.0567	0.0578	0.0620	0.0713	0.9832	0.9982
0.6	0.0402	0.0304	0.0257	0.0219	0.0179	0.0145	0.0546	0.0533	0.0554	0.0569	0.0613	0.0695	0.9838	0.9982
0.7	0.0402	0.0304	0.0252	0.0215	0.0178	0.0143	0.0543	0.0532	0.0545	0.0561	0.0610	0.0688	0.9840	0.9981
0.8	0.0402	0.0276	0.0252	0.0215	0.0172	0.0137	0.0546	0.0510	0.0541	0.0552	0.0593	0.0669	0.9847	0.9982
0.9	0.0304	0.0276	0.0246	0.0217	0.0169	0.0130	0.0437	0.0467	0.0490	0.0508	0.0542	0.0609	0.9855	0.9982
1	0.0013	0.0015	0.0013	0.0012	0.0013	0.0013	0.0013	0.0018	0.0020	0.0022	0.0029	0.0042	0.9986	0.9995



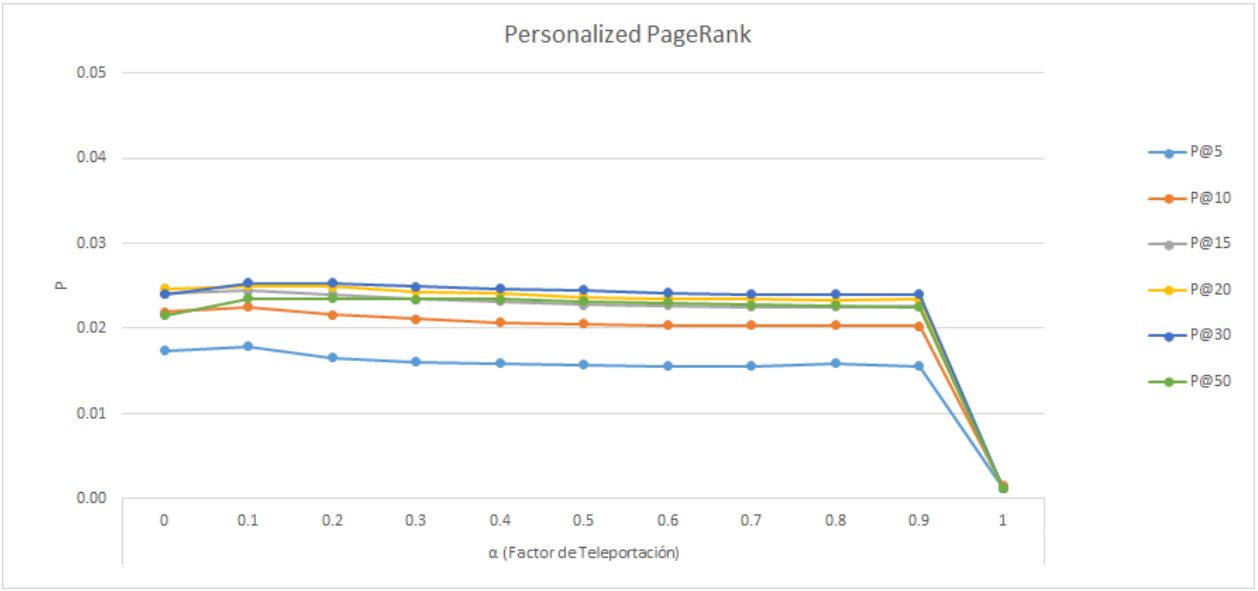

PageRank
0,030
0,025
0,020
0,015
0,010
0,005
0,000
P
1,000
0,995
0,990
0,985
0,980
0,975
0,970
EPC, ILD
P@50
EPC@50
ILD@50
0
0,1
0,2
0,3
0,4
0,5
0,6
0,7
0,8
0,9
1
α (Factor de Teleportación)

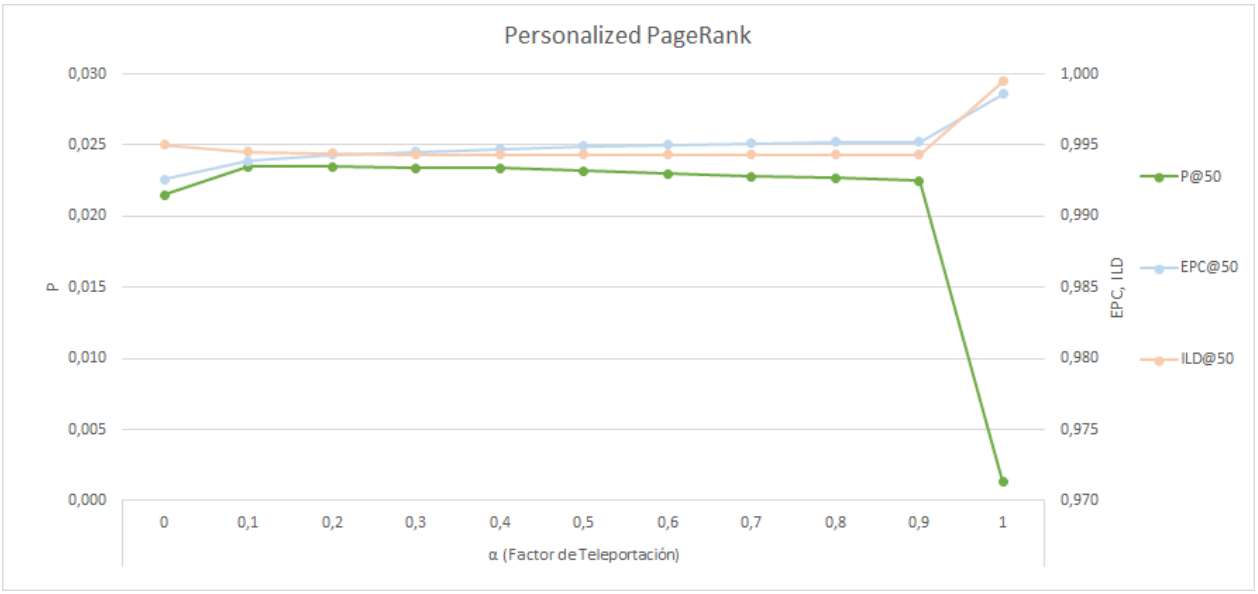

PageRank
0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0.00
nDCG
nDCG@5
nDCG@10
nDCG@15
nDCG@20
nDCG@30
nDCG@50
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
α (Factor de Teleportación)

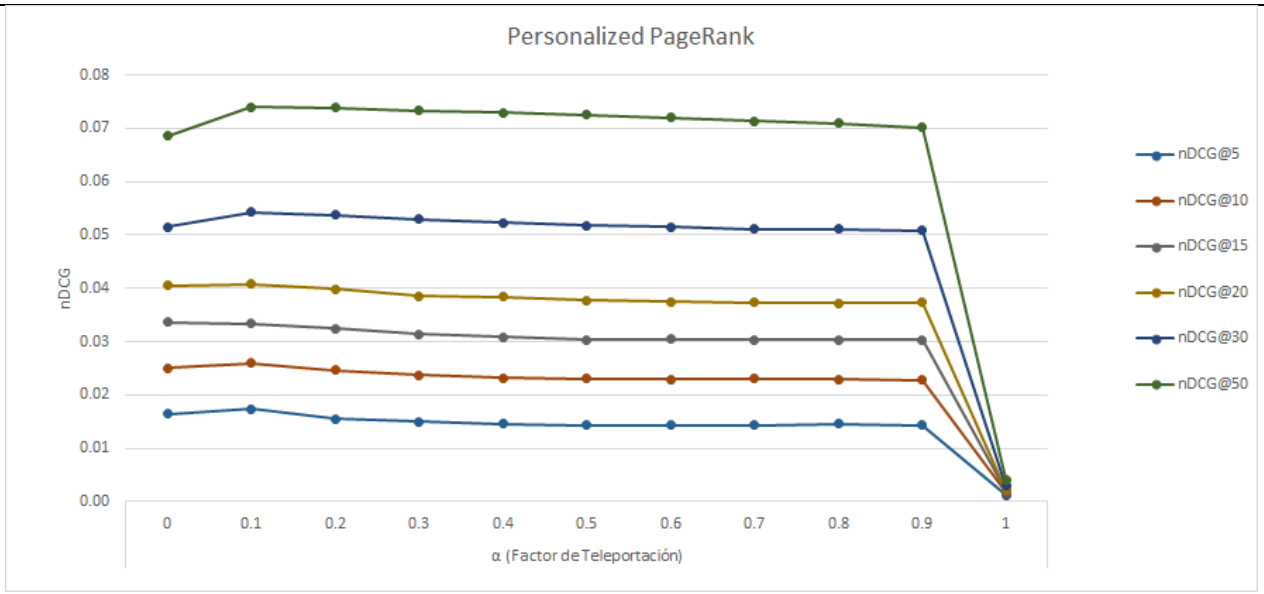

PageRank
0,08
0,07
0,06
0,05
0,04
0,03
0,02
0,01
0,00
nDCG
1,000
0,995
0,990
0,985
0,980
0,975
0,970
EPC, ILD
nDCG@50
EPC@50
ILD@50
0
0,1
0,2
0,3
0,4
0,5
0,6
0,7
0,8
0,9
1
α (Factor de Teleportación)

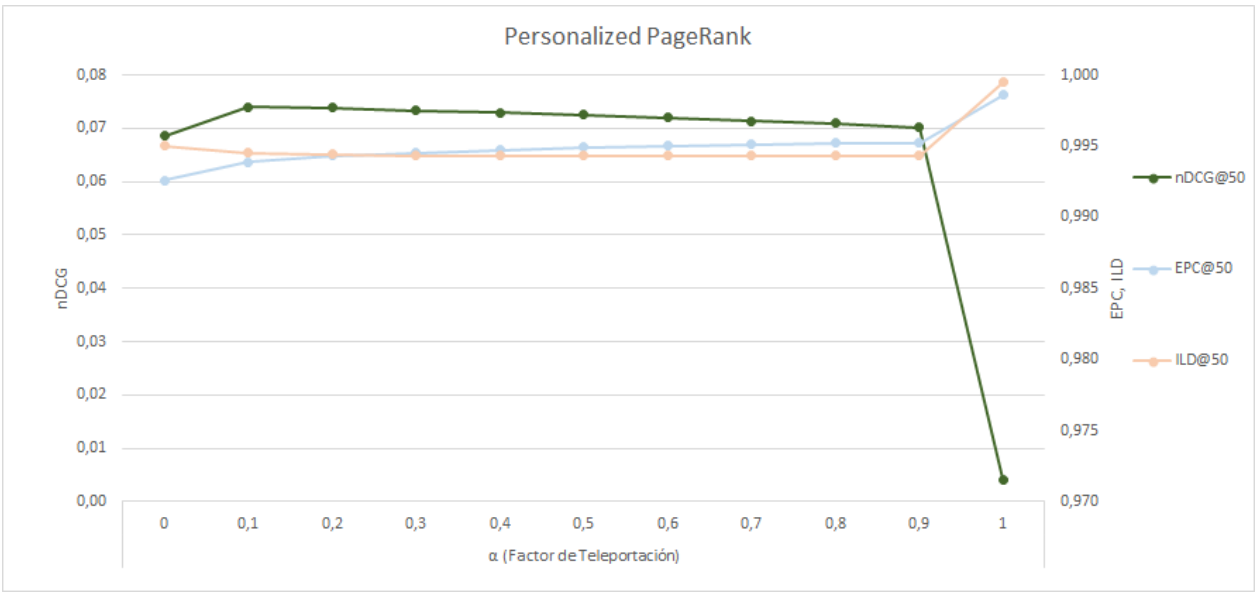
4.2.2 Efecto en las evaluaciones de la variación del factor de teleportación en el algoritmo PageRank Personalizado

α (Factor de Teleportación)	P@5	P@10	P@15	P@20	P@30	P@50	nDCG@5	nDCG@10	nDCG@15	nDCG@20	nDCG@30	nDCG@50	EPC@50	ILD@50
0	0.0174	0.0219	0.0241	0.0247	0.0240	0.0215	0.0165	0.0251	0.0336	0.0405	0.0515	0.0686	0.9926	0.9950
0.1	0.0179	0.0225	0.0244	0.0250	0.0253	0.0235	0.0174	0.0260	0.0334	0.0408	0.0543	0.0740	0.9939	0.9945
0.2	0.0166	0.0216	0.0240	0.0249	0.0253	0.0235	0.0155	0.0246	0.0324	0.0398	0.0538	0.0738	0.9943	0.9944
0.3	0.0161	0.0211	0.0234	0.0243	0.0249	0.0234	0.0150	0.0237	0.0314	0.0386	0.0529	0.0733	0.9945	0.9943
0.4	0.0159	0.0207	0.0232	0.0241	0.0247	0.0234	0.0146	0.0232	0.0309	0.0384	0.0523	0.0729	0.9947	0.9943
0.5	0.0157	0.0205	0.0228	0.0237	0.0245	0.0232	0.0144	0.0230	0.0304	0.0377	0.0518	0.0725	0.9949	0.9943
0.6	0.0156	0.0204	0.0227	0.0235	0.0242	0.0230	0.0143	0.0229	0.0305	0.0375	0.0515	0.0720	0.9950	0.9943
0.7	0.0156	0.0204	0.0225	0.0235	0.0240	0.0228	0.0144	0.0230	0.0303	0.0374	0.0511	0.0714	0.9951	0.9943
0.8	0.0159	0.0204	0.0225	0.0233	0.0240	0.0227	0.0146	0.0229	0.0304	0.0372	0.0511	0.0709	0.9952	0.9943
0.9	0.0156	0.0203	0.0226	0.0234	0.0240	0.0225	0.0144	0.0228	0.0304	0.0374	0.0509	0.0702	0.9952	0.9943
1	0.0013	0.0015	0.0013	0.0012	0.0013	0.0013	0.0013	0.0018	0.0020	0.0022	0.0029	0.0042	0.9986	0.9995



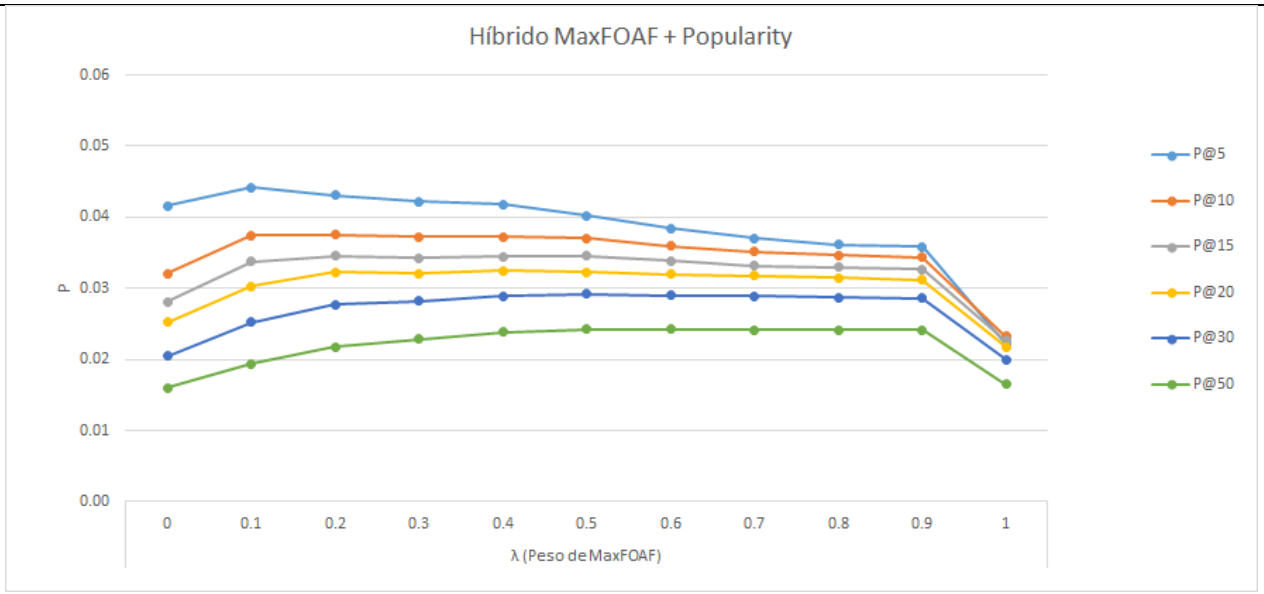


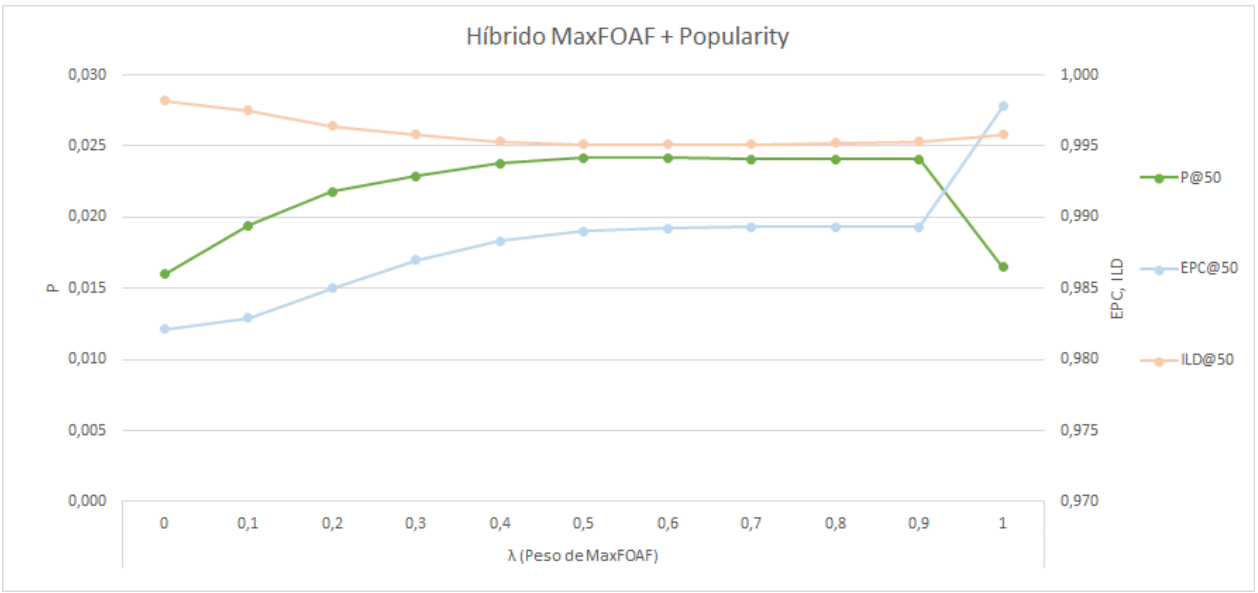


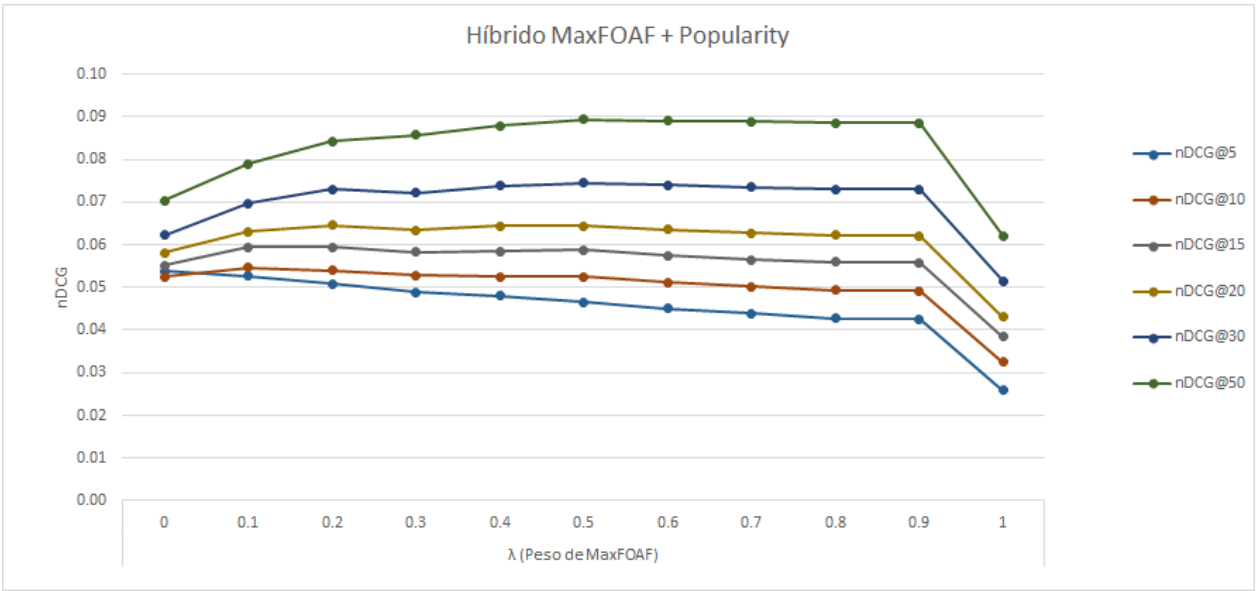


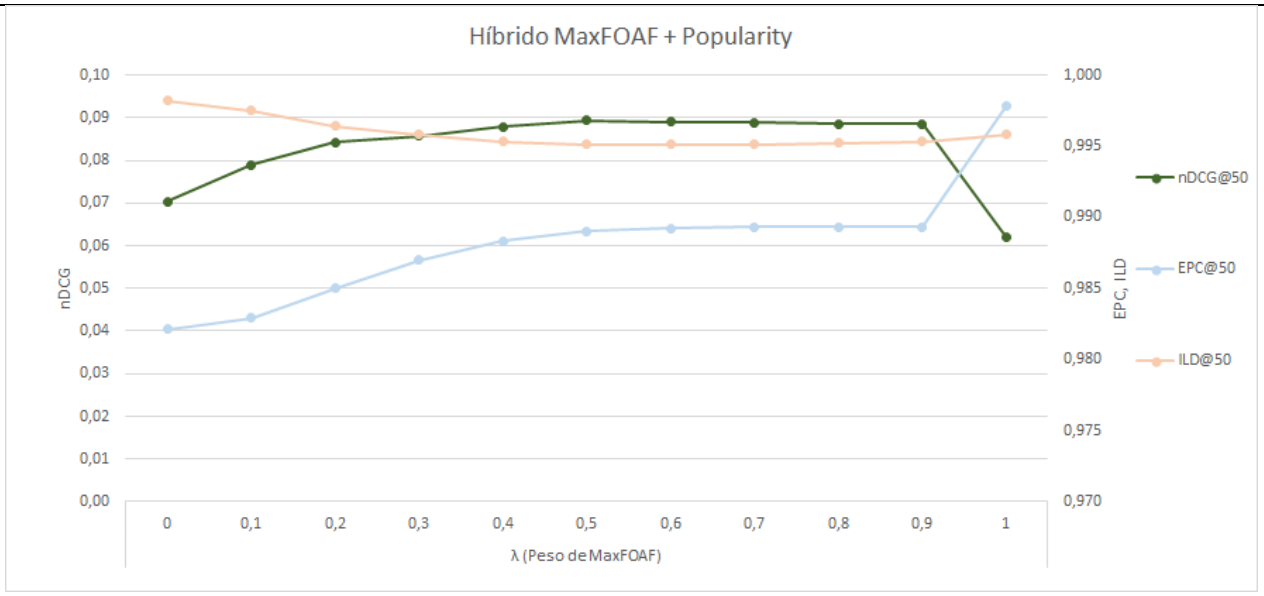
4.2.3 Efecto en las evaluaciones de la variación del peso de cada algoritmo en el algoritmo híbrido MaxFOAF + Popularity

λ (Peso de MaxFOAF)	P@5	P@10	P@15	P@20	P@30	P@50	nDCG@5	nDCG@10	nDCG@15	nDCG@20	nDCG@30	nDCG@50	EPC@50	ILD@50
0	0.0416	0.0321	0.0281	0.0252	0.0205	0.0160	0.0538	0.0525	0.0551	0.0581	0.0622	0.0704	0.9821	0.9982
0.1	0.0442	0.0374	0.0338	0.0303	0.0252	0.0194	0.0527	0.0547	0.0594	0.0631	0.0696	0.0789	0.9829	0.9975
0.2	0.0431	0.0375	0.0346	0.0323	0.0277	0.0218	0.0508	0.0540	0.0595	0.0646	0.0729	0.0842	0.9850	0.9964
0.3	0.0422	0.0372	0.0343	0.0321	0.0282	0.0229	0.0489	0.0528	0.0582	0.0634	0.0722	0.0857	0.9870	0.9958
0.4	0.0418	0.0372	0.0345	0.0325	0.0289	0.0238	0.0480	0.0524	0.0584	0.0643	0.0738	0.0878	0.9883	0.9953
0.5	0.0402	0.0370	0.0346	0.0323	0.0292	0.0242	0.0466	0.0525	0.0587	0.0643	0.0745	0.0893	0.9890	0.9951
0.6	0.0384	0.0360	0.0339	0.0320	0.0290	0.0242	0.0450	0.0512	0.0574	0.0635	0.0740	0.0891	0.9892	0.9951
0.7	0.0370	0.0352	0.0332	0.0318	0.0289	0.0241	0.0438	0.0502	0.0565	0.0628	0.0735	0.0888	0.9893	0.9951
0.8	0.0361	0.0347	0.0330	0.0315	0.0287	0.0241	0.0428	0.0494	0.0560	0.0622	0.0730	0.0886	0.9893	0.9952
0.9	0.0359	0.0344	0.0327	0.0312	0.0286	0.0241	0.0426	0.0491	0.0558	0.0620	0.0729	0.0885	0.9893	0.9953
1	0.0222	0.0233	0.0227	0.0218	0.0200	0.0165	0.0258	0.0325	0.0384	0.0431	0.0515	0.0621	0.9978	0.9958


Híbrido MaxFOAF + Popularity
P
λ (Peso de MaxFOAF)
P@5
P@10
P@15
P@20
P@30
P@50

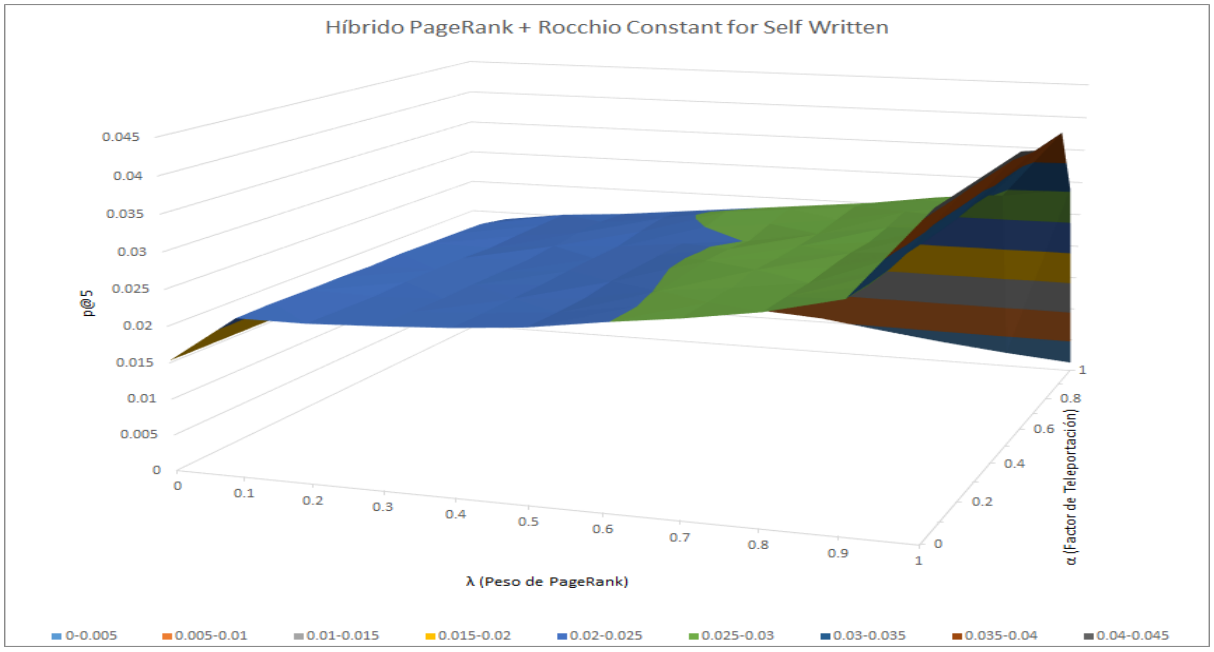

Híbrido MaxFOAF + Popularity
P
EPC, ILD
λ (Peso de MaxFOAF)
P@50
EPC@50
ILD@50


Híbrido MaxFOAF + Popularity
nDCG
λ (Peso de MaxFOAF)
nDCG@5
nDCG@10
nDCG@15
nDCG@20
nDCG@30
nDCG@50



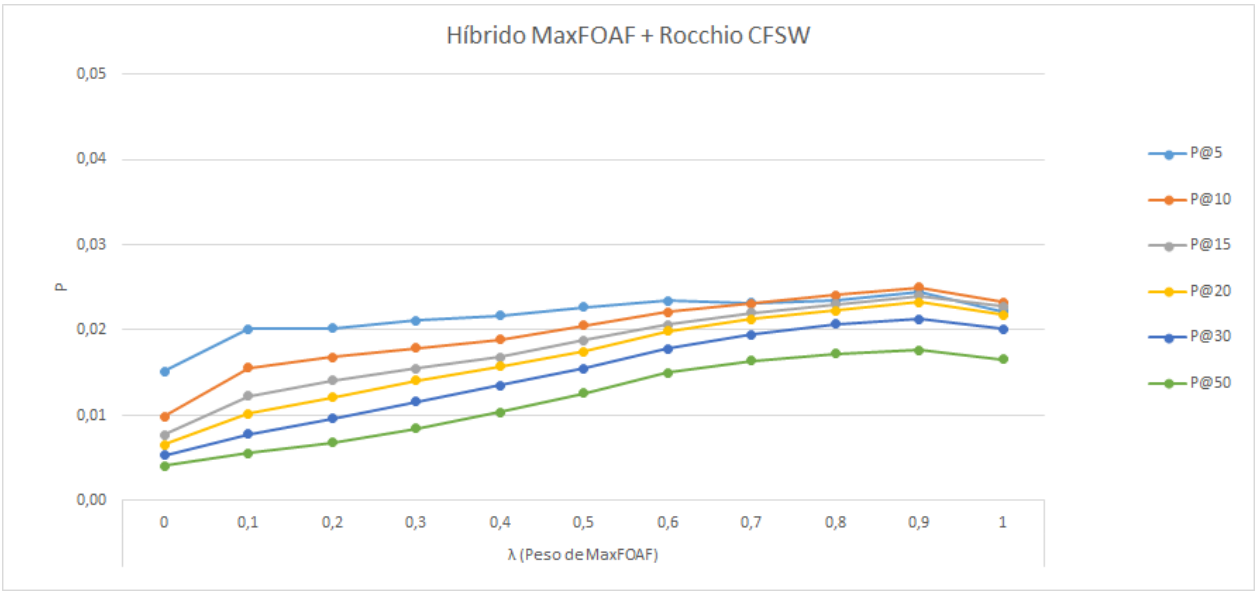
4.2.4 Efecto en la evaluación de la métrica P@5 de la variación del peso de cada algoritmo y el factor de teleportación en el algoritmo híbrido PageRank + Rocchio CSFW

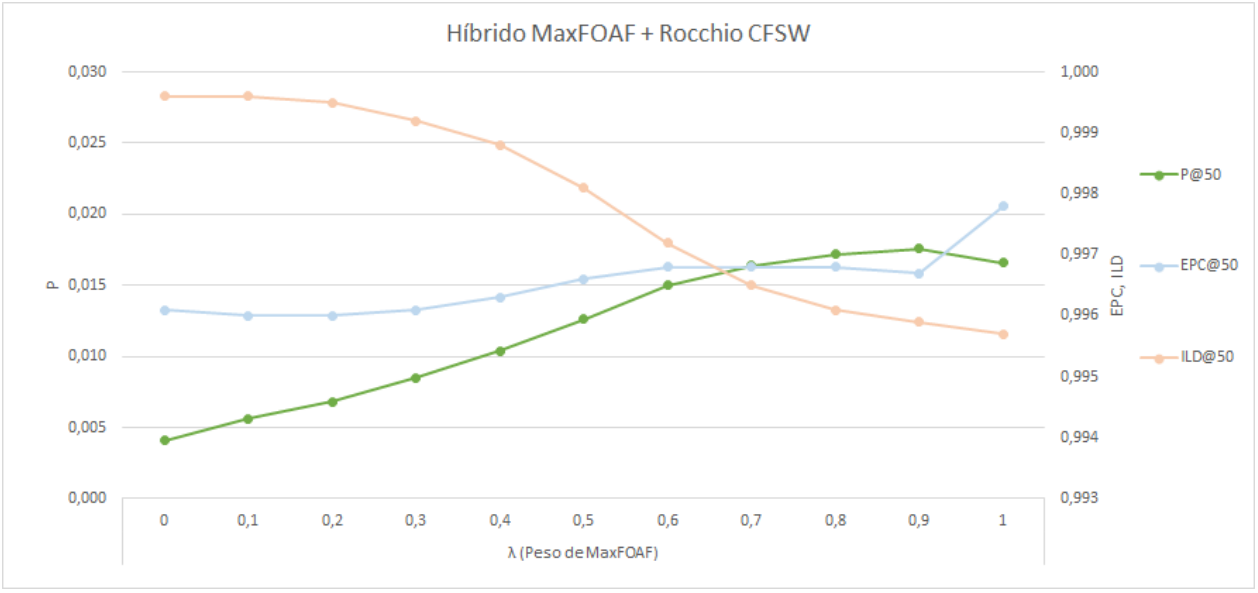
α (Factor de Teleportación) \ λ (Peso de PageRank)	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
0	0.0153	0.0218	0.0218	0.0222	0.0227	0.0235	0.0249	0.0261	0.0276	0.0299	0.0414
0.1	0.0153	0.0219	0.0217	0.0221	0.0226	0.0234	0.0249	0.0262	0.0276	0.0296	0.0414
0.2	0.0153	0.0218	0.0216	0.022	0.0227	0.0234	0.0251	0.0262	0.0275	0.0297	0.0414
0.3	0.0153	0.0218	0.0218	0.0221	0.0227	0.0239	0.0254	0.0268	0.0278	0.0303	0.0414
0.4	0.0153	0.0217	0.0218	0.0221	0.0229	0.024	0.0257	0.0266	0.028	0.0302	0.0414
0.5	0.0153	0.0218	0.0218	0.0221	0.023	0.0241	0.0256	0.0266	0.0278	0.0302	0.0414
0.6	0.0153	0.0217	0.0216	0.0225	0.023	0.0241	0.0251	0.0267	0.0283	0.0303	0.0402
0.7	0.0153	0.0216	0.0218	0.0229	0.0237	0.025	0.026	0.0272	0.029	0.0306	0.0402
0.8	0.0153	0.0215	0.0219	0.023	0.0242	0.0257	0.0267	0.0276	0.0292	0.0305	0.0402
0.9	0.0153	0.0207	0.022	0.0227	0.0237	0.0247	0.0257	0.0267	0.0281	0.0289	0.0304
1	0.0153	0.0124	0.0111	0.0083	0.0071	0.0062	0.0056	0.0043	0.0031	0.0021	0.0013

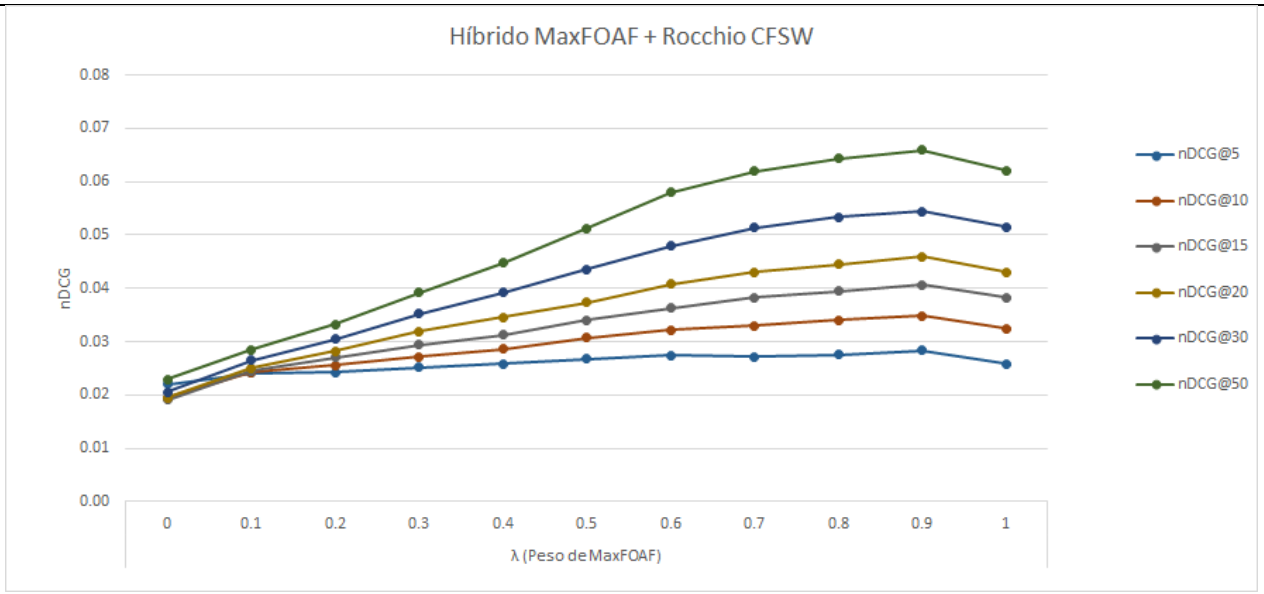


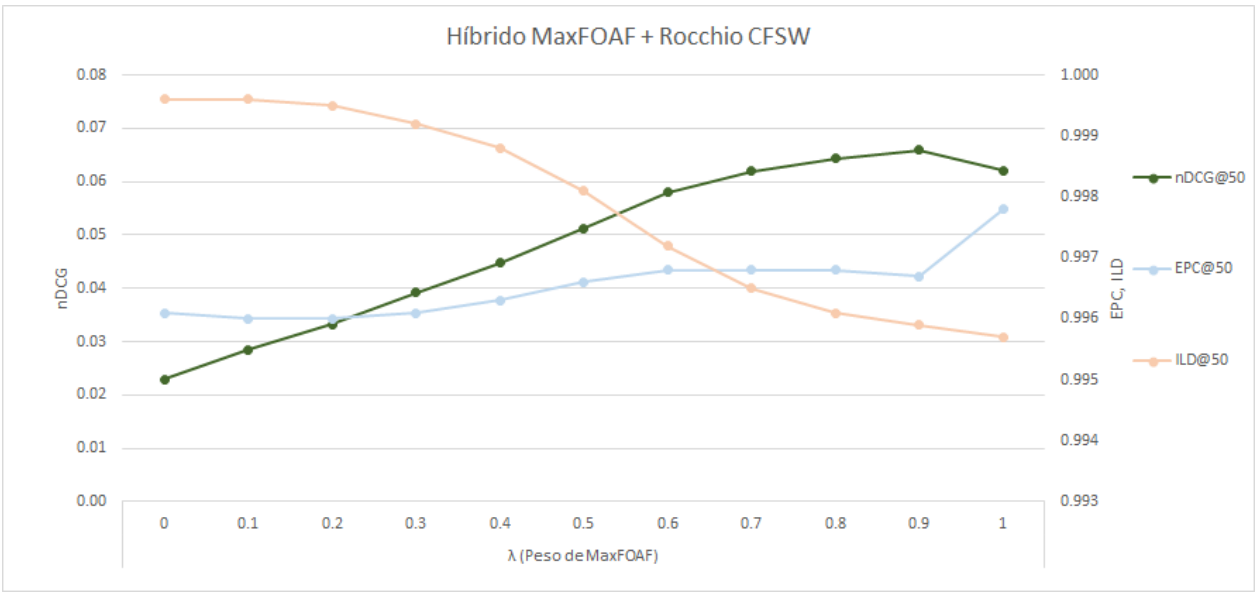
4.2.5 Efecto en las evaluaciones de la variación del peso de cada en el algoritmo híbrido MaxFOAF + Rocchio CSFW

λ (Peso de MaxFOAF)	P@5	P@10	P@15	P@20	P@30	P@50	nDCG@5	nDCG@10	nDCG@15	nDCG@20	nDCG@30	nDCG@50	EPC@50	ILD@50
0	0.0152	0.0099	0.0077	0.0066	0.0053	0.0041	0.0220	0.0195	0.0191	0.0195	0.0206	0.0229	0.9961	0.9996
0.1	0.0201	0.0156	0.0123	0.0102	0.0078	0.0056	0.0241	0.0242	0.0245	0.0250	0.0265	0.0285	0.9960	0.9996
0.2	0.0202	0.0168	0.0141	0.0121	0.0096	0.0068	0.0242	0.0256	0.0270	0.0282	0.0305	0.0332	0.9960	0.9995
0.3	0.0211	0.0179	0.0155	0.0141	0.0116	0.0085	0.0252	0.0272	0.0294	0.0319	0.0353	0.0392	0.9961	0.9992
0.4	0.0217	0.0189	0.0168	0.0157	0.0135	0.0104	0.0259	0.0286	0.0313	0.0346	0.0392	0.0447	0.9963	0.9988
0.5	0.0227	0.0205	0.0188	0.0175	0.0155	0.0126	0.0268	0.0307	0.0341	0.0374	0.0436	0.0513	0.9966	0.9981
0.6	0.0234	0.0221	0.0206	0.0199	0.0178	0.0150	0.0274	0.0322	0.0363	0.0408	0.0480	0.0580	0.9968	0.9972
0.7	0.0232	0.0231	0.0220	0.0213	0.0195	0.0164	0.0272	0.0330	0.0383	0.0430	0.0514	0.0620	0.9968	0.9965
0.8	0.0234	0.0241	0.0230	0.0223	0.0207	0.0172	0.0275	0.0340	0.0395	0.0445	0.0533	0.0644	0.9968	0.9961
0.9	0.0245	0.0250	0.0240	0.0233	0.0213	0.0176	0.0283	0.0348	0.0406	0.0460	0.0544	0.0659	0.9967	0.9959
1	0.0222	0.0233	0.0228	0.0218	0.0201	0.0166	0.0258	0.0324	0.0383	0.0431	0.0515	0.0621	0.9978	0.9957



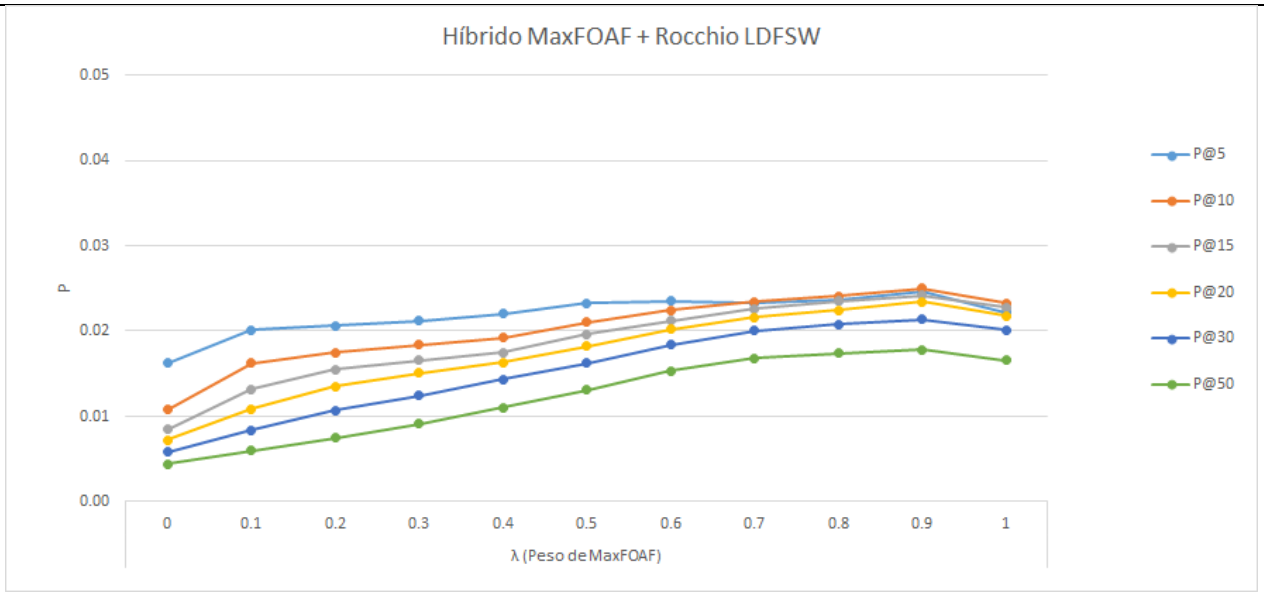


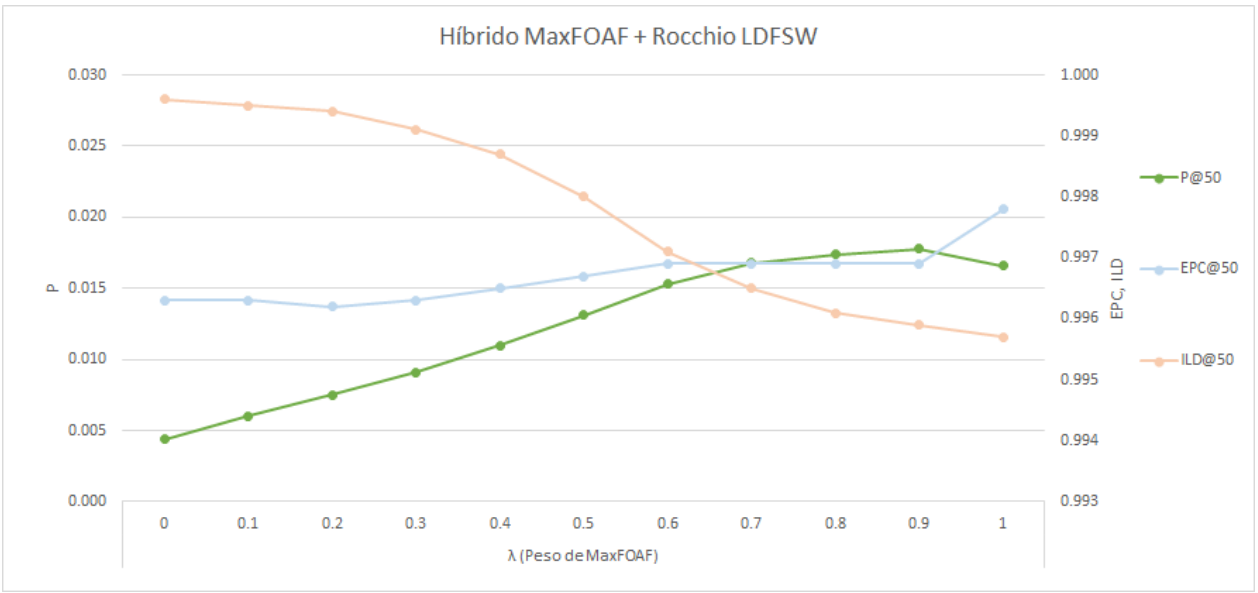


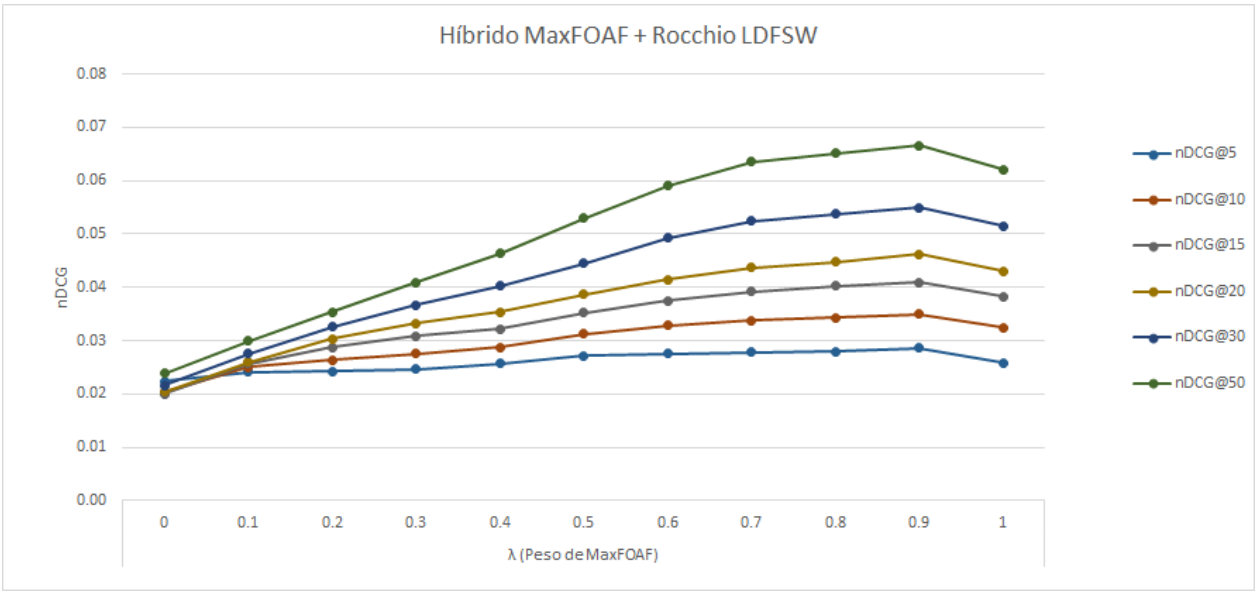


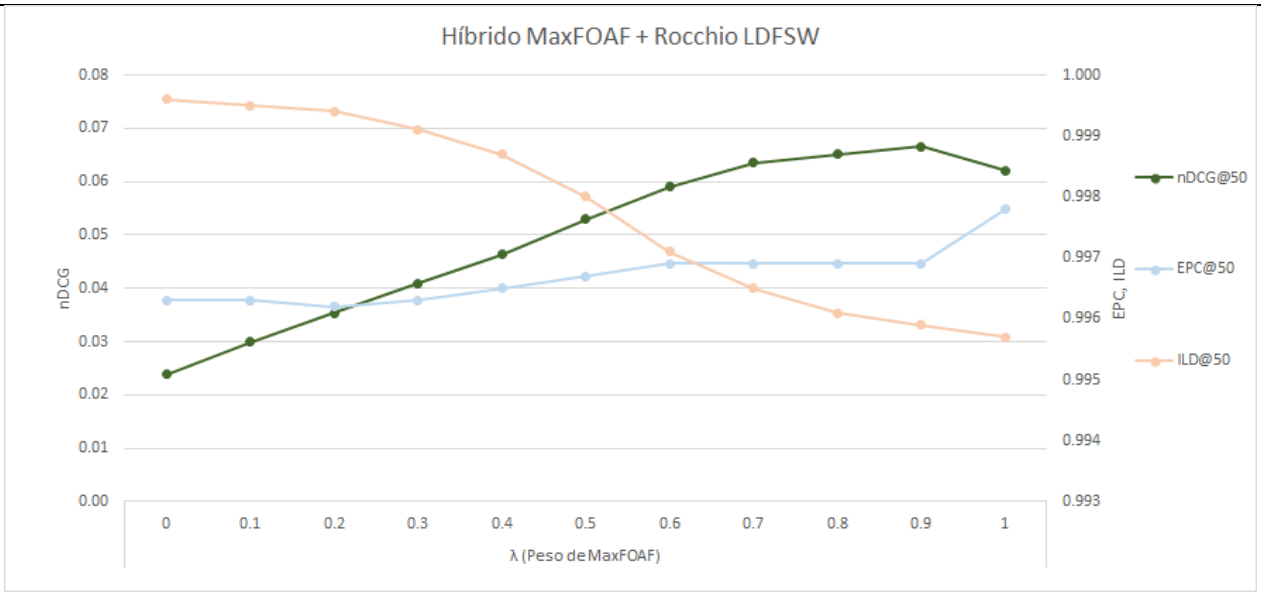
4.2.6 Efecto en las evaluaciones de la variación del peso de cada en el algoritmo híbrido MaxFOAF + Rocchio LDFSW

λ (Peso de MaxFOAF)	P@5	P@10	P@15	P@20	P@30	P@50	nDCG@5	nDCG@10	nDCG@15	nDCG@20	nDCG@30	nDCG@50	EPC@50	ILD@50
0	0.0162	0.0108	0.0085	0.0072	0.0058	0.0044	0.0224	0.0203	0.0200	0.0204	0.0216	0.0239	0.9963	0.9996
0.1	0.0201	0.0162	0.0132	0.0109	0.0084	0.0060	0.0241	0.0250	0.0257	0.0260	0.0276	0.0299	0.9963	0.9995
0.2	0.0206	0.0175	0.0155	0.0135	0.0107	0.0075	0.0243	0.0264	0.0287	0.0304	0.0326	0.0354	0.9962	0.9994
0.3	0.0212	0.0184	0.0166	0.0151	0.0124	0.0091	0.0246	0.0276	0.0309	0.0332	0.0367	0.0409	0.9963	0.9991
0.4	0.0220	0.0192	0.0175	0.0163	0.0143	0.0110	0.0257	0.0288	0.0322	0.0354	0.0402	0.0463	0.9965	0.9987
0.5	0.0233	0.0210	0.0196	0.0182	0.0162	0.0131	0.0272	0.0313	0.0352	0.0387	0.0445	0.0530	0.9967	0.9980
0.6	0.0235	0.0224	0.0212	0.0202	0.0184	0.0153	0.0276	0.0328	0.0375	0.0415	0.0493	0.0590	0.9969	0.9971
0.7	0.0233	0.0234	0.0226	0.0216	0.0200	0.0168	0.0278	0.0338	0.0392	0.0437	0.0524	0.0635	0.9969	0.9965
0.8	0.0236	0.0241	0.0235	0.0224	0.0208	0.0174	0.0279	0.0343	0.0402	0.0448	0.0538	0.0651	0.9969	0.9961
0.9	0.0246	0.0250	0.0242	0.0234	0.0214	0.0178	0.0286	0.0350	0.0409	0.0462	0.0550	0.0666	0.9969	0.9959
1	0.0222	0.0233	0.0228	0.0218	0.0201	0.0166	0.0258	0.0324	0.0383	0.0431	0.0515	0.0621	0.9978	0.9957


Híbrido MaxFOAF + Rocchio LDFSW
P
λ (Peso de MaxFOAF)
P@5
P@10
P@15
P@20
P@30
P@50

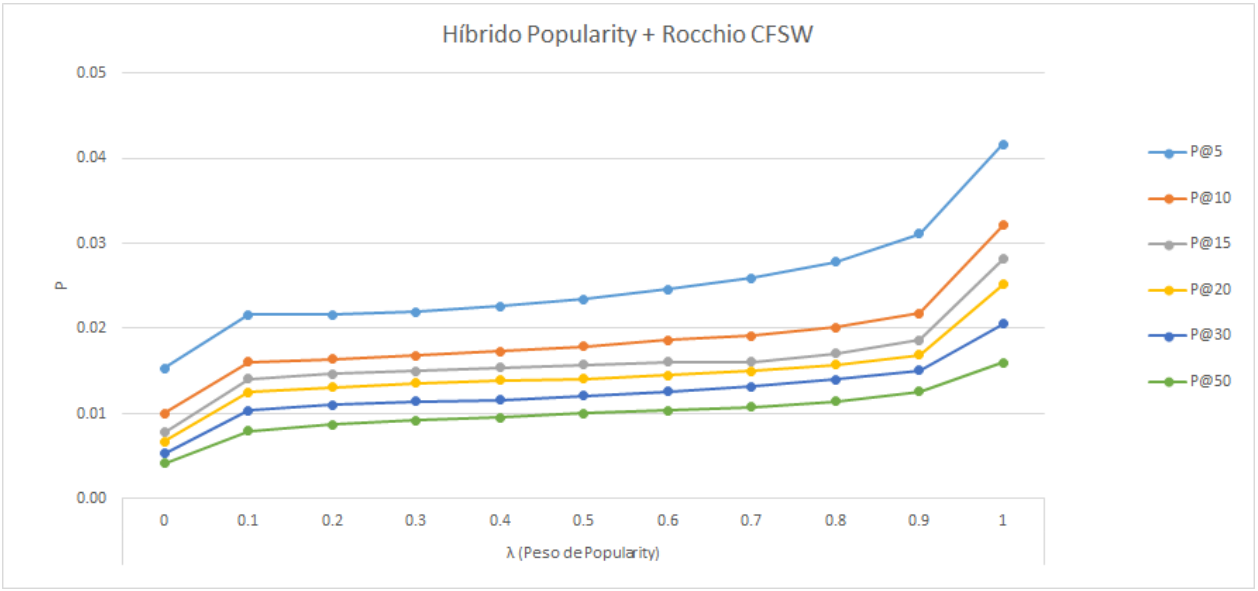

Híbrido MaxFOAF + Rocchio LDFSW
P
EPC, ILD
λ (Peso de MaxFOAF)
P@50
EPC@50
ILD@50

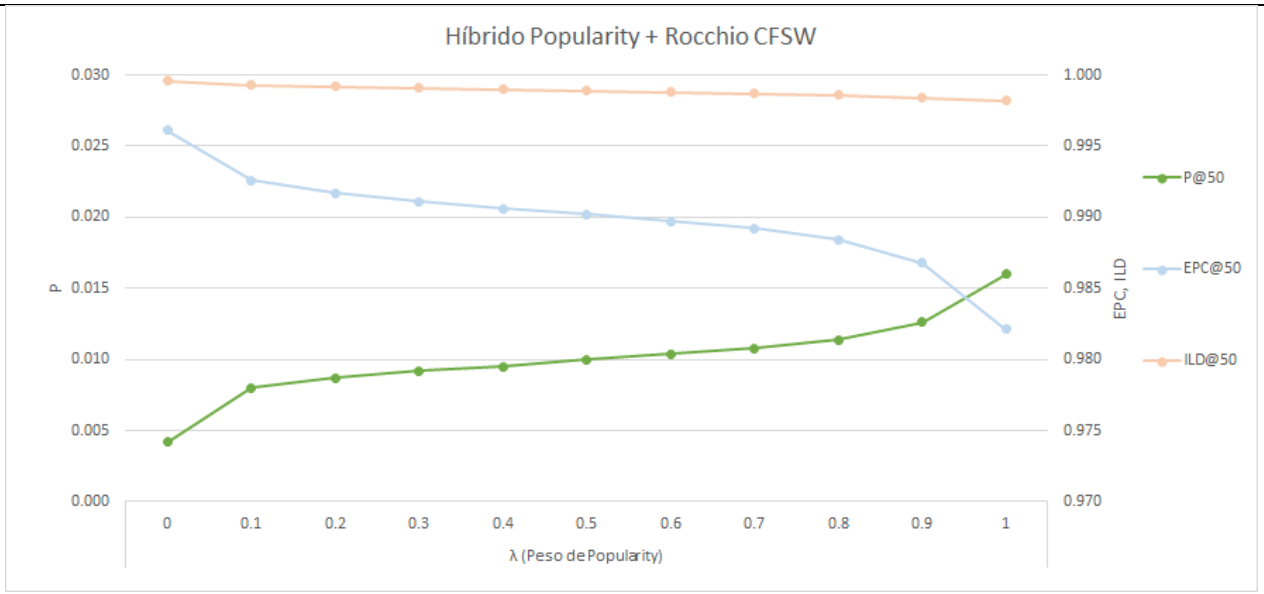

Híbrido MaxFOAF + Rocchio LDFSW
nDCG
λ (Peso de MaxFOAF)
nDCG@5
nDCG@10
nDCG@15
nDCG@20
nDCG@30
nDCG@50

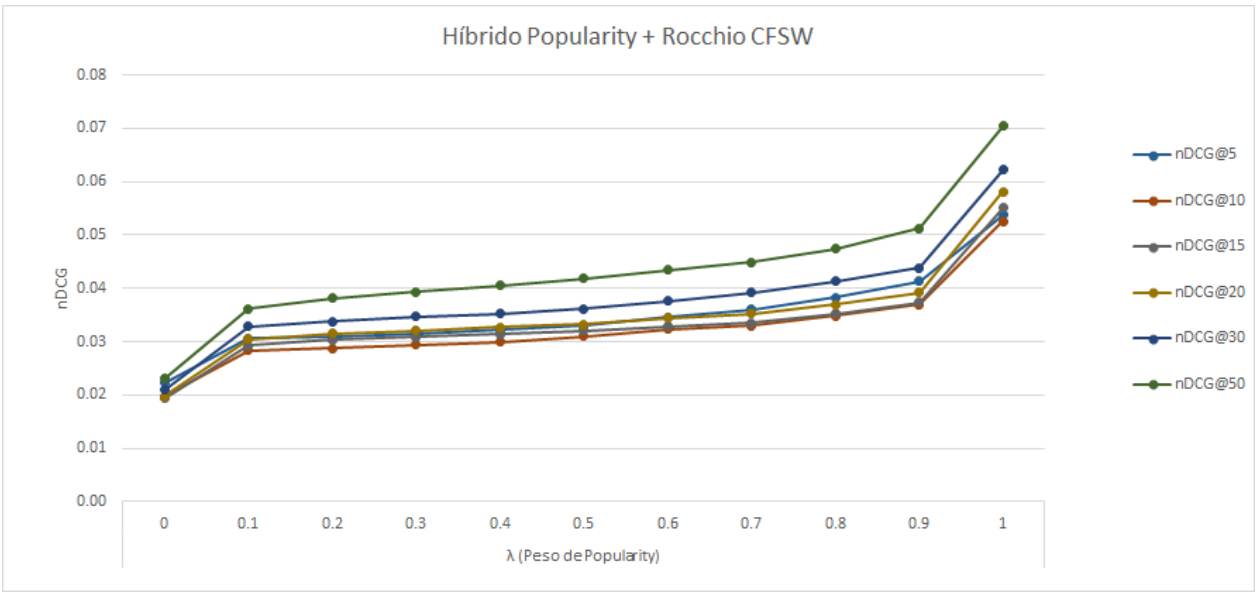


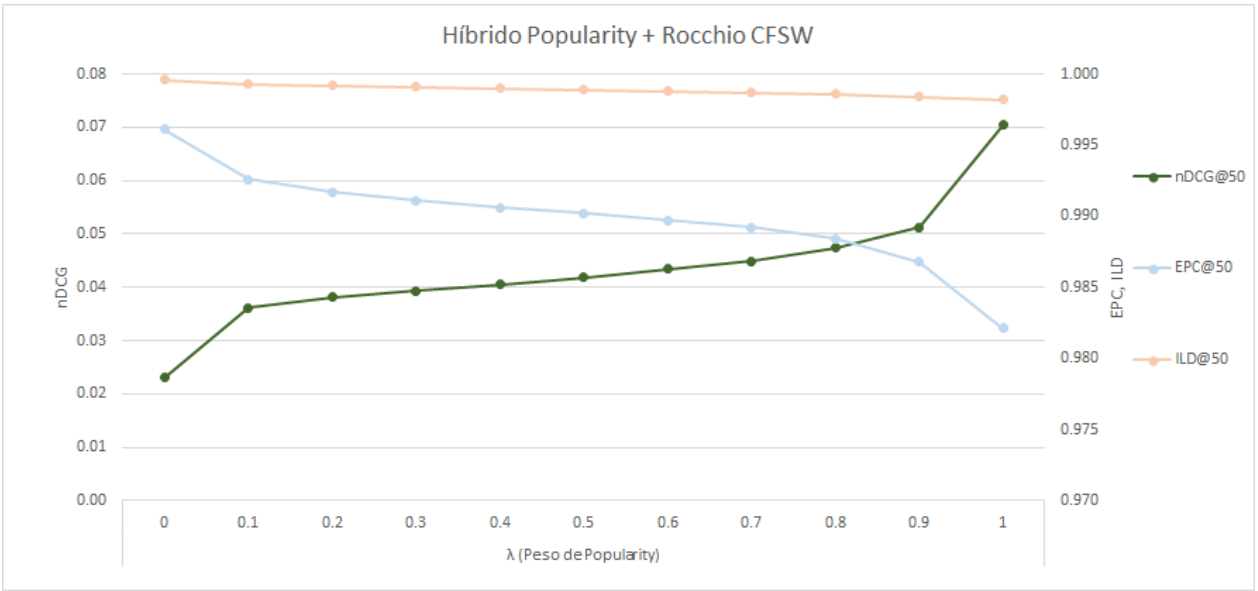
4.2.7 Efecto en las evaluaciones de la variación del peso de cada en el algoritmo híbrido Popularity + Rocchio CFSW

λ (Peso de Popularity)	P@5	P@10	P@15	P@20	P@30	P@50	nDCG@5	nDCG@10	nDCG@15	nDCG@20	nDCG@30	nDCG@50	EPC@50	ILD@50
0	0.0153	0.0100	0.0078	0.0067	0.0053	0.0042	0.0223	0.0197	0.0194	0.0198	0.0209	0.0231	0.9961	0.9996
0.1	0.0216	0.0161	0.0141	0.0125	0.0104	0.0080	0.0306	0.0283	0.0293	0.0305	0.0328	0.0362	0.9926	0.9993
0.2	0.0216	0.0164	0.0147	0.0131	0.0110	0.0087	0.0309	0.0288	0.0303	0.0315	0.0338	0.0381	0.9917	0.9992
0.3	0.0219	0.0168	0.0150	0.0136	0.0114	0.0092	0.0315	0.0294	0.0309	0.0321	0.0347	0.0394	0.9911	0.9991
0.4	0.0226	0.0173	0.0154	0.0139	0.0116	0.0095	0.0322	0.0300	0.0314	0.0327	0.0352	0.0405	0.9906	0.9990
0.5	0.0234	0.0179	0.0157	0.0141	0.0121	0.0100	0.0330	0.0310	0.0320	0.0333	0.0361	0.0418	0.9902	0.9989
0.6	0.0246	0.0186	0.0161	0.0145	0.0126	0.0104	0.0346	0.0323	0.0329	0.0344	0.0376	0.0435	0.9897	0.9988
0.7	0.0259	0.0191	0.0161	0.0150	0.0132	0.0108	0.0360	0.0330	0.0335	0.0352	0.0392	0.0449	0.9892	0.9987
0.8	0.0278	0.0201	0.0171	0.0157	0.0140	0.0114	0.0383	0.0348	0.0353	0.0370	0.0413	0.0474	0.9884	0.9986
0.9	0.0311	0.0218	0.0186	0.0169	0.0151	0.0126	0.0413	0.0370	0.0373	0.0392	0.0438	0.0512	0.9868	0.9984
1	0.0416	0.0321	0.0281	0.0252	0.0205	0.0160	0.0538	0.0525	0.0551	0.0581	0.0622	0.0704	0.9821	0.9982



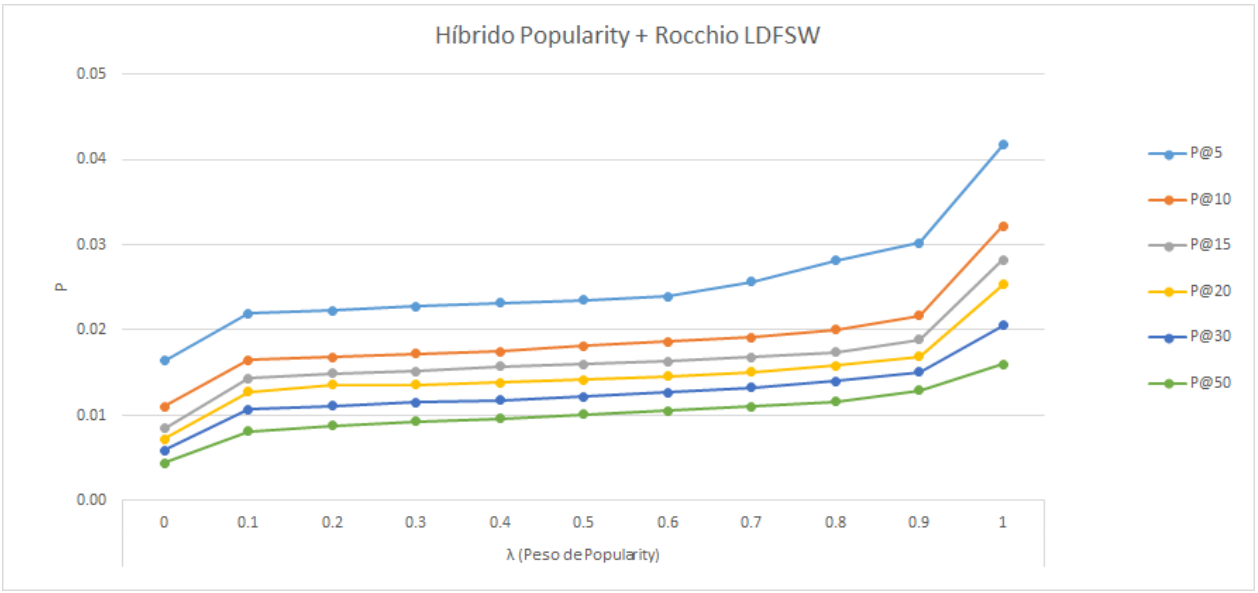

Híbrido Popularity + Rocchio CFSW
P
EPC, ILD
λ (Peso de Popularity)
P@50
EPC@50
ILD@50

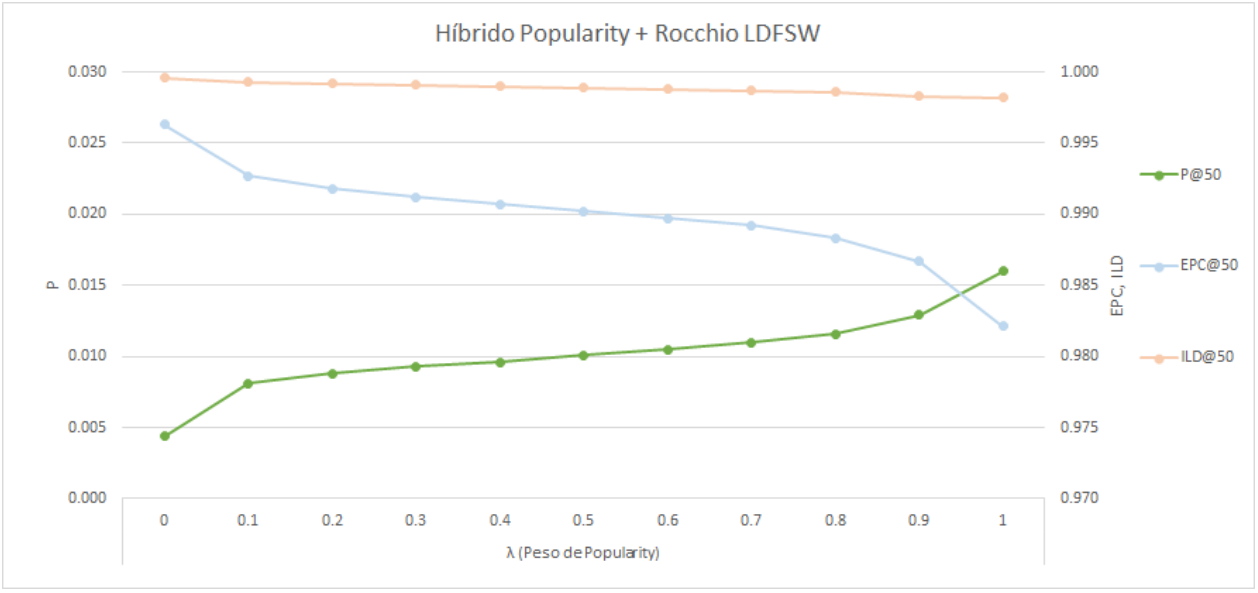

Híbrido Popularity + Rocchio CFSW
nDCG
λ (Peso de Popularity)
nDCG@5
nDCG@10
nDCG@15
nDCG@20
nDCG@30
nDCG@50

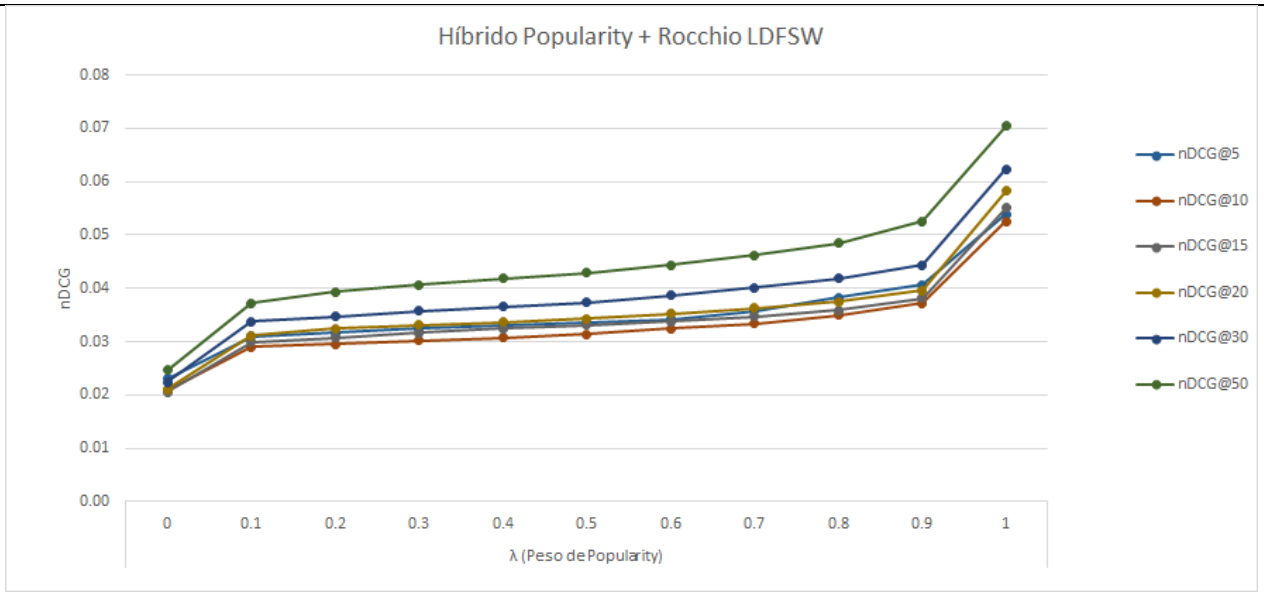

Híbrido Popularity + Rocchio CFSW
nDCG
EPC, ILD
λ (Peso de Popularity)
nDCG@50
EPC@50
ILD@50

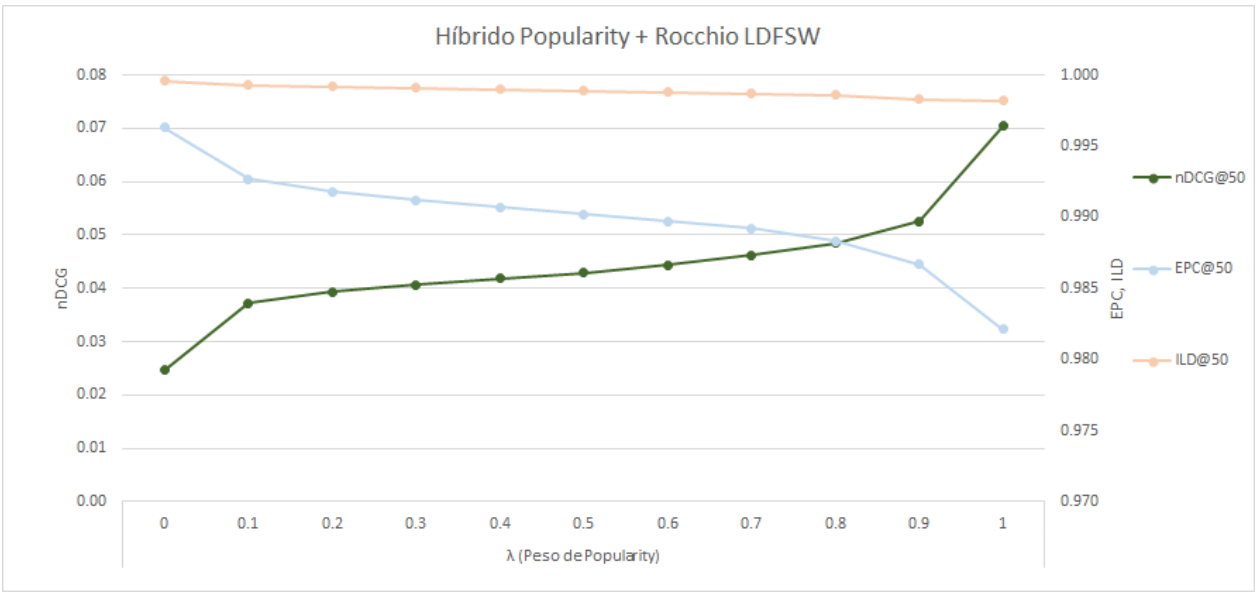
4.2.8 Efecto en las evaluaciones de la variación del peso de cada en el algoritmo híbrido Popularity + Rocchio LDFSW

λ (Peso de Popularity)	P@5	P@10	P@15	P@20	P@30	P@50	nDCG@5	nDCG@10	nDCG@15	nDCG@20	nDCG@30	nDCG@50	EPC@50	ILD@50
0	0.0164	0.0110	0.0085	0.0072	0.0059	0.0044	0.0230	0.0210	0.0206	0.0211	0.0224	0.0246	0.9963	0.9996
0.1	0.0219	0.0165	0.0143	0.0128	0.0107	0.0081	0.0310	0.0290	0.0299	0.0312	0.0338	0.0372	0.9927	0.9993
0.2	0.0223	0.0168	0.0149	0.0136	0.0111	0.0088	0.0316	0.0295	0.0308	0.0325	0.0347	0.0393	0.9918	0.9992
0.3	0.0228	0.0172	0.0152	0.0136	0.0115	0.0093	0.0324	0.0302	0.0317	0.0331	0.0357	0.0407	0.9912	0.9991
0.4	0.0232	0.0175	0.0157	0.0138	0.0118	0.0096	0.0330	0.0307	0.0325	0.0337	0.0365	0.0418	0.9907	0.9990
0.5	0.0235	0.0181	0.0160	0.0142	0.0122	0.0101	0.0335	0.0314	0.0331	0.0343	0.0374	0.0429	0.9902	0.9989
0.6	0.0239	0.0186	0.0163	0.0146	0.0127	0.0105	0.0342	0.0324	0.0338	0.0352	0.0387	0.0444	0.9897	0.9988
0.7	0.0257	0.0191	0.0168	0.0151	0.0133	0.0110	0.0358	0.0334	0.0347	0.0363	0.0401	0.0462	0.9892	0.9987
0.8	0.0281	0.0200	0.0174	0.0158	0.0140	0.0116	0.0383	0.0350	0.0359	0.0375	0.0418	0.0485	0.9883	0.9986
0.9	0.0302	0.0217	0.0189	0.0169	0.0151	0.0129	0.0407	0.0372	0.0382	0.0396	0.0444	0.0525	0.9867	0.9983
1	0.0417	0.0322	0.0282	0.0253	0.0205	0.0160	0.0539	0.0526	0.0551	0.0582	0.0623	0.0704	0.9821	0.9982






Híbrido Popularity + Rocchio LDFSW
nDCG
0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0.00
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
λ (Peso de Popularity)
nDCG@5
nDCG@10
nDCG@15
nDCG@20
nDCG@30
nDCG@50


Híbrido Popularity + Rocchio LDFSW
nDCG
EPC, ILD
1.000
0.995
0.990
0.985
0.980
0.975
0.970
λ (Peso de Popularity)
nDCG@50
EPC@50
ILD@50